T.C.
ANKARA MÜZİK VE GÜZEL SANATLAR ÜNİVERSİTESİ
MÜZİK VE GÜZEL SANATLAR ENSTİTÜSÜ

PyxelWidgets: MÜZİKAL IZGARA KONTROLCÜLER İÇİN GRAFİKSEL KULLANICI ARAYÜZÜ ÇATISI TASARIMI

YÜKSEK LİSANS TEZİ

Malik Enes ŞAFAK

Müzik Teknolojileri Anabilim Dalı
Müzik Teknolojileri Tezli Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. Abdurrahman TARİKCİ

Temmuz 2022

T.C.
ANKARA MÜZİK VE GÜZEL SANATLAR ÜNİVERSİTESİ
MÜZİK VE GÜZEL SANATLAR ENSTİTÜSÜ

PyxelWidgets: MÜZİKAL IZGARA KONTROLCÜLER İÇİN GRAFİKSEL KULLANICI ARAYÜZÜ ÇATISI TASARIMI

YÜKSEK LİSANS TEZİ

Malik Enes ŞAFAK
2002035

Müzik Teknolojileri Anabilim Dalı
Müzik Teknolojileri Tezli Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. Abdurrahman TARİKCİ

Temmuz 2022

TEZ ONAYI

MGÜ Müzik ve Güzel Sanatlar Enstitüsü'nün 2002035 numaralı Yüksek Lisans Öğrencisi Malik Enes ŞAFAK ilgili yönetmeliklerce belirlenmiş gerekli tüm şartları yerine getirmek suretiyle PyxelWidgets: MÜZİKAL IZGARA KONTROLCÜLER İÇİN GRAFİKSEL KULLANICI ARAYÜZÜ ÇATISI TASARIMI başlıklı tezini hazırlamış ve aşağıda imzaları bulunan jüri önünde, başarıyla sunmuştur.

Tez Danışmanı	Prof. Dr. Abdurrahman TARİKCİ Ankara Müzik ve Güzel Sanatlar Üniversitesi	_____
Jüri Üyeleri	Prof. Dr. Arda EDEN Yıldız Teknik Üniversitesi	_____
	Doç. Dr. Hasan DELEN Ankara Müzik ve Güzel Sanatlar Üniversitesi	_____

Teslim Tarihi :
Savunma Tarihi : 28 07 2022

ETİK BEYAN

Ankara Müzik ve Güzel Sanatlar Üniversitesi Müzik ve Güzel Sanatlar Enstitüsü Lisansüstü Tez Yazım Kılavuzunda belirtilen hususlar doğrultusunda hazırladığım bu tez çalışmasının akademik ve etik kurallar çerçevesinde gerçekleştirilmiş olduğunu; tezde yer verdiğim tüm bilgi, belge, bulgu, yorum ve sonuçları bilimsel etik ilkelere bağlı kalarak oluşturduğumu; yararlandığım eserlerin tümüne etik kurallar dâhilinde atıfta bulunup kaynakçada yer verdiğimi ve Enstitüye teslim ettiğim çalışmanın bütünüyle özgün nitelikte olduğunu bildirir; tez çalışmamla ilgili olarak burada dile getirdiğim kişisel bildirimin aksi yönünde ortaya çıkabilecek ve etik kuralların ihlal edilmiş olduğuna işaret eden bir tespit durumunda, akademik kariyer bağlamında aleyhime doğabilecek tüm hak kayıplarını peşinen kabullendiğimi beyan ederim.

21.07.2022 Malik Enes ŞAFAK

ÖNSÖZ

Müzikal ızgara kontrolcüler özellikle elektronik müzik üretiminde yaygın olarak kullanılan araçlardır. Uygun yazılım desteği kullanım amacına göre özelleştirilebilme potansiyeline sahip olan bu araçlar için verilen yazılım desteğinin sınırlı olduğu görülmüştür. Bu çalışma, müzikal ızgara kontrolcülerin kullanıcıların ihtiyaçların göre özelleştirilerek daha verimli bir çalışma ortamının oluşturulmasına yardımcı olmak amacıyla gerçekleştirilmiştir.

Bu çalışmanın yürütülmesinde ve yazımında yapmış olduğu katkılardan dolayı tez danışmanım sayın Prof. Dr. Abdurrahman TARİKCİ'ye, mesleğimin ve bu tezin şekillenmesine katkılarından dolayı sayın Prof. Dr. Arda EDEN'e, tezin yazım sürecinde desteklerini esirgemeyen Doğuhan Eren KARACAN'a, çalışma arkadaşlarım Serkan ÇOLAK'a, Uğur BALOĞLU'na ve aileme teşekkürlerimi sunarım.

Temmuz 2022 Malik Enes ŞAFAK

İÇİNDEKİLER

Sayfa

KISALTMALAR

USI	: Universal Synthesizer Interface
MIDI	: Musical Instruments Digital Interface
AES	: Audio Engineering Society
NAMM	: National Association of Music Merchants
DAW	: Digital Audio Workstation
GUI	: Graphical User Interface
LED	: Light Emitting Diode

ŞEKİL LİSTESİ

Sayfa

PyxelWidgets: MÜZİKAL IZGARA KONTROLCÜLER İÇİN GRAFİKSEL KULLANICI ARAYÜZÜ ÇATISI TASARIMI

ÖZET

Bu tezde, müzikal ızgara kontrolcülere yeni kontrol parametreleri eklemek için Python programlama dili kullanılarak PyxelWidgets adında bir grafiksel kullanıcı arayüzü çatısı geliştirilmiştir. Müzikal ızgara kontrolcüler tarafından MIDI protokolünün yaygın olarak kullanılması sebebi ile PyxelWidgets çatısı tarafından desteklenen donanım ve yazılımlar ile iletişim için MIDI 1.0 protokolü kullanılmıştır.
İlk olarak müzikal ızgara kontrolcülere diğer müzikal kontrolcüler üzerindeki hangi ögelerin eklenebileceği incelenmiştir. Bu amaca ulaşmak için müzikal kontrolcüler üzerindeki kontroller performans ögeleri ve kontrol ögeleri olarak iki kategoriye ayrılmıştır. Daha sonrasında bu ögeler incelenerek müzikal ızgara kontrolcü üzerinde nasıl ifade edilebilecekleri analiz edilmiştir. Benzer şekilde, müzikal ızgara kontrolcülerin de renk bilgisini nasıl aldıkları ve kullanıcı girişini nasıl ilettikleri analiz edilmiştir. Daha sonrasında PyxelWidgets analiz edilen müzikal ızgara kontrolcülerden ve kontrol ögelerinden elde edilen veriler kullanılarak geliştirilmiştir. PyxelWidgets çatısı farklı iletişim yöntemlerini kullanan bütün müzikal ızgara kontrolcüler ile çalışması için tasarlanmıştır. PyxelWidgets'ın çalışmasını kontrol etmek için Akai APC40 MK2 ve Novation Launchpad MK3 adlı iki farklı müzikal ızgara kontrolcüsü ile test edilmiş ve eklenen her yeni kontrolcü ile başarıyla çalıştığı gözlemlenmiştir. Ayrıca, PyxelWidgets gelecekte ortaya çıkacak farklı müzikal ızgara kontrolcülere de uyarlanabilecek bir mimari ile geliştirilmiştir. Birden çok pencerenin ve birden çok müzikal ızgara kontrolcünün bir arada kullanılmasını sağlayan bir pencere yöneticisi de geliştirilerek kullanıcı deneyiminin arttırılması hedeflenmiştir.
Müzikal ızgara kontrolcülerin, PyxelWidgets'in özellikleri ve yapısı sayesinde, sahip olduğu özelliklerin genişletilmesi, kullanıcıların ses tasarımında ve prodüksiyonunda kullandıkları donanımları ve yazılımları kontrol etmek için daha çok olanağa sahip olmasını sağlamaktadır. Bu geliştirmelerin sonucu olarak müzisyenler, ses mühendisleri ve ses tasarımcıları müzikal ızgara kontrolcüleri ile sahip oldukları ekipmanları kolaylıkla kontrol edebileceklerdir. Ayrıca kullanım amacına uygun olarak özelleştirilmiş müzikal ızgara kontrolcüler, kullanıcının daha hızlı işlem yapmasını sağlayarak, müzik prodüksiyonu için daha çok zamana sahip olmasını sağlayacaktır. PyxelWidgets çatısı kullanılarak geliştirilen grafiksel kullanıcı arayüzleri ile müzikal ızgara kontrolcülerin kapasitelerinin artmasından dolayı, ekipman çeşitliliği gereksiniminde de azalma olasılığı ortaya çıkmaktadır.

Anahtar Kelimeler

Müzikal Izgara Kontrolcüleri, MIDI, Grafiksel Kullanıcı Arayüzü Çatısı, Nesne Yönelimli Programlama, Python

PyxelWidgets: GRAPHICAL USER INTERFACE FRAMEWORK DESIGN FOR MUSICAL GRID CONTROLLERS

SUMMARY

In this thesis, in order to add new control parameters to musical grid controllers, graphical user interface framework, called as PyxelWidgets, is developed by using Python programming language. MIDI 1.0 protocol is used to communicate hardwares or softwares that are supported by PyxelWidgets, since musical grid controllers communicate with MIDI.

First, it is examined to find which control parameters can be added to musical grid controllers.To achieve this aim, controllers are grouped into two categories named as control elements and performance elements. Then it is analyzed to determine how to express these elements. Similarly, musical grid controllers are analyzed to find how they get color information and how to transmit user inputs. After that, PyxelWidgets is developed using the results obtained by analysis of the musical grid controllers. In fact, PyxelWidgets is designed such a way that it can be used with all kinds of musical grid controllers. To check PyxelWidgets performance, it is tested in two models of musical grid controllers, Akai APC40 MK2 and Novation Launchpad MK3, and seen that all kinds of new controllers added work successfully. Moreover, by means of the architecture of PyxelWidgets, it can be adapted to new musical grid controllers that will emerge in the future. To improve user-friendliness, window manager is developed to use more than one windows or musical grid controllers.

Through properties and structure of the PyxelWidgets, musical grid controllers capacity improves and users get more ability to control their hardwares or softwares, which are used in sound design and production. As a result of these improvements, musicians, sound engineers, and sound designers can more easily use their equipment that can be used with musical grid controllers. Furthermore, they have more time while production due to faster processing. Moreover, due to the increase in the capacities of the musical grid controllers, the possibility of a decrease in the need for equipment diversity arises.

Keywords

Musical Grid Controllers, MIDI, Graphical User Interface Framework, Object Oriented Programing, Python

1. GİRİŞ

Müzik de ses gibi üretimi, yayılması ve algılanması olarak üç aşamada değerlendirilebilir. Üretim aşamasında çalgılar, insan sesi ve benzeri araçlar kullanılmaktadır. Yayılması esnasında çalgıların biçimleri ve sahip olduğu teknik özellikler önem kazanmaktadır. Algılanmasında ise kimi insanların işitme yeteneklerinin yanı sıra mikrofon gibi araçlar da algılayıcı olarak düşünülebilir. Bütün bu yöntemler ve araçlar teknoloji ile beraber değişim göstermektedir.

Bu değişimleri en belirgin biçimde akustik ve elektronik müzik enstrümanlarında gözlemlemek mümkündür. Akustik müzik enstrümanları; ses üretim mekanizması, ergonomi, çalınabilirlik, icracının ifade şekli ve estetik arasındaki dengeyi sağlayabilmek için yüzlerce yıl boyunca form değişikliklerine uğramış ve günümüzdeki şeklini almıştır. Elektronik müzik enstrümanları ise yüz yıldan biraz daha uzun zaman önce hayatımıza girmiş olmalarına rağmen teknolojinin gelişimi ile akustik müzik enstrümanları ile benzer ölçekte ses üretim ve kontrol mekanizmalarında değişim meydana gelmiştir (Paradiso ve O'Modhrain, 2003).

Elektronik müzik enstrümanlarındaki bu değişim, dijital teknolojilerin hayatımıza girmesiyle birlikte yeni bir ivme kazanmıştır. Bu gelişmelerin ardından ortaya çıkmış olan enstrümanların kendi aralarında iletişim kurma gereği ortaya çıkmıştır. Kimi durumlarda diğer elektronik malzemeler ile de iletişim kurulması gerekebilmektedir. Dijital müzik enstrümanlarının haberleşmesi için tasarlanan müzik enstrümanları dijital arabirimi (musical instruments digital interface, MIDI) teknolojisi, bu çalgıların ses üretim mekanizmaları ile kontrol mekanizmaları arasındaki ayrımın belirginleşmesini sağlamıştır. Hatta MIDI teknolojisi, icracının hareketlerini yeni ve sıra dışı yollarla yakalayarak ses üretim mekanizmasına aktaran alternatif müzikal kontrolcülerin gelişimini desteklemiştir (Jordà, 2020). Bu kontrolcüler çoğunlukla MIDI kontrolcüsü olarak anılmaktadır.

Bilgisayarların işlem gücü arttıkça yazılımsal müzik enstrümanlarının kullanımı yaygınlaşmaya başlamıştır. Bilgisayarlar aracılığıyla canlı performans sırasında icracının ihtiyaç duyabileceği bütün parametreler görülebilmekte ve değiştirilebilmektedir. Fakat hem bilgisayarların üzerindeki klavye, dokunmatik

yüzey, fare gibi araçların müzikal ifade için yetersiz olması, hem de icracının bilgisayar ile etkileşiminin dinleyici ile arasındaki iletişimi azaltması (Vallis, Hochenbaum ve Kapur, 2010) sebebiyle icracılar, ihtiyaç duydukları özellikleri barındıran özel müzikal kontrolcülere gereksinim duymuştur. Fakat icracıya özel müzikal kontrolcü tasarımı hem yüksek maliyetli bir yöntemdir hem de kontrolcünün tek bir icracının istekleri doğrultusunda tasarlanmış olması, başka icracılar tarafından kullanılmasını zorlaştırmaktadır. İcracıya özel müzikal kontrolcü tasarımı problemine başka bir yaklaşım ise müzikal ızgara kontrolcülerdir (Vallis ve diğerleri, 2010).

Müzikal ızgara kontrolcüler, ortogonal biçimde dizilmiş hücre adı verilen ve bilgi aktarımı ya da görsel geri bildirim için kullanılan kısımlardan oluşan müzikal kontrolcülerdir. Daha açık bir ifadeyle, bu hücreler hem kullanıcının hücre ile etkileşimini bağlı olduğu cihaza iletebilmekte, hem de kendi üzerindeki LED'lerin göstereceği renk değerlerini bağlı olduğu cihazdan alabilmektedir. Uygun yazılım desteği ile birlikte, ızgara kontrolcüler üzerinde, müzikal kontrolcülerde yaygın olarak kullanılan ögeler taklit edilebilmektedir. Bu sayede müzikal ızgara kontrolcüler tamamen özelleştirilmiş bir müzikal kontrolcü deneyimi sunabilmektedirler (Rossmy ve Wiethoff, 2021; Vallis ve diğerleri, 2010).

1.1. Problem Durumu

Müzikal ızgara kontrolcüler için sunulan yazılım desteği üretici şirket tarafından, kontrolcünün birlikte kullanılacağı yazılım tarafından ya da üçüncü parti yazılımlar tarafından sağlanabilmektedir. Örnek olarak şu yazılımlar sunulabilir:

1. Novation Components (Novation Digital Music Systems Ltd, t.y.-a) yazılımı Novation şirketinin Launchpad serisi cihazlarından 3. nesil müzikal ızgara kontrolcülerine özelleştirilmiş arayüzler hazırlamayı mümkün kılmaktadır. Cihaz içerisine bir kere yüklenen bu arayüzün kullanılması için ek bir yazılım kullanılması gerekmemektedir. Fakat bu yazılımın hem kısıtlı sayıda cihaza yönelik olduğu, hem de eklenen araçların boyutlarının ve işlevlerinin kısıtlı özelleştirme seçeneğine sahip olduğu gözlemlenmiştir.
2. Ableton şirketinin Live yazılımı için geliştirilmiş Launchpad95 (Henri, 2022) yazılımı, açık kaynaklı müzikal ızgara kontrolcü arayüzüne örnek olarak gösterilebilir. Launchpad95 yazılımı, oldukça fazla özelliğe sahip olsa da yalnızca Live programı içerisinde çalışmaktadır ve kısıtlı sayıda cihazı

desteklemektedir. Aynı zamanda oluşturulmuş arayüzün kişiselleştirilmesi, ancak yazılımın kodu üzerinde büyük değişiklikler yapılarak mümkün olmaktadır.

Gösterilen örnek yazılımlarda görülebildiği gibi sağlanan destek ya tek bir cihaz ile ya da tek bir DAW ile birlikte çalışacak şekilde sunulmaktadır. Bu durum gelişmiş özellikler isteyen kullanıcıları ya belirli cihazları ya da belirli yazılımları kullanmaya mecbur bırakmaktadır. Öte yandan mevcut destek yazılımları, kullanıcılara istedikleri araçları özelleştirme imkânını sunmakta oldukça kısıtlıdır. Dolayısıyla bütün müzikal ızgara kontrolcüler tarafından ortak olarak kullanılabilecek ve kolaylıkla özelleştirilebilecek yazılımlara ihtiyaç olduğu saptanmıştır.

1.2. PyxelWidgets Yaklaşımı

Bu çalışmada farklı müzikal ızgara kontrolcülerin ortak yazılımsal araçlar kullanılarak özelleştirilebilmesi için Python (Rossum ve Drake, 2010) programlama dilinde bir grafiksel kullanıcı arayüzü (graphical user interface, GUI) çatısı (framework) geliştirilmiştir. Bu çatıya PyxelWidgets adı verilmiştir. PyxelWidgets farklı iletişim protokollerini ve yöntemlerini kullanan bütün müzikal ızgara kontrolcüler ile çalışacak şekilde tasarlanmıştır. Geliştirilen çatı ile birlikte, kullanıcıların kullanma alışkanlıklarını sürdürebilmeleri için, müzikal kontrolcülerde yaygın olarak kullanılanlara benzer biçimde tasarlanmış grafiksel araçlar da geliştirilmesi amaçlanmıştır.

Müzikal ızgara kontrolcüleri kendi çalışma ortamlarına göre özelleştirmek isteyen geliştiriciler PyxelWidgets çatısını kullanarak yeni arayüzler geliştirebilmektedirler. Çatı, arayüze eklenen araçların durumlarının değişmesi halinde geliştiriciye haber vermektedir. Bu sayede geliştirici durumu değişen aracın hangi işlevi yerine getireceğini belirleyebilmektedir. Örneğin ızgara kontrolcüye eklenen bir sürgülü direnç (fader) aracının kullanıcı tarafından ilgili hücrelere basılarak durumunun değiştirilmesi halinde geliştirici bu bilgiyi dijital ses istasyonuna (digital audio workstation, DAW) ileterek bir kanalın ses seviyesinin değiştirilmesini ya da odanın ışık seviyesinin kontrol edilmesini sağlayabilmektedir. PyxelWidgets tarafından hesaplanan değerin hangi işlevi yerine getireceği tamamen geliştiricinin inisiyatifindedir.

PyxelWidgets çatısı kullanılarak bir arayüz geliştirilebilmesi için Python programlama dilinde uygun yazılımın geliştirilmesi gerekmektedir. Pyxelwidgets çatısı, diğer yazılımlara kıyasla daha zorlayıcı bir öğrenme eğrisine sahip olsa da çok daha fazla özelleştirme seçeneğine sahiptir. Ayrıca PyxelWidgets çatısı kullanılarak geliştirilmiş bir yazılımın çok küçük değişiklikler ile farklı müzikal ızgara kontrolcüler ile kullanılması da mümkündür.

2. KAVRAMSAL ÇERÇEVE VE LİTERATÜR TARAMASI

2.1. Müzikal Kontrolcüler

Modern teknoloji, müzikal enstrümanların kontrol mekanizmaları ile ses üretim mekanizmalarının ayrılmasını sağlamıştır. Örneğin, nefesli akustik müzik enstrümanlarında, eklenen bağlantı parçaları icracının aksi halde erişemeyeceği uzaklıktaki boşluklara erişmesini sağlayarak enstrümanın ses aralığının genişlemesini mümkün kılmıştır. Kontrol mekanizması açısından en gelişmiş akustik müzik enstrümanlarından birinin borulu org olduğu kabul edilebilir. Borulu orgların kontrol mekanizması, çoğunlukla tahta üflemeli borulardan oluşan boru setlerinin icracı tarafından birden çok klavye ile kontrol edebilmesini sağlayabilmektedir. Bu sayede icracının fiziksel olarak ses üretim mekanizması ile etkileşime geçmesi gerekmemektedir. Borulu orglarda genellikle org klavyesinin hangi boru setlerini kontrol edeceğini belirlemek de mümkündür. Böylece icracı tek bir kontrolcü üzerinden birden çok sesi aynı anda duyurabilmektedir. Müzikal enstrümanların ses üretim mekanizmaları ile kontrol mekanizmalarının ayrılması bu iki kısmın ayrı ayrı optimize edilmesine de izin vermektedir. Bu sayede müzikal kontrol mekanizması icracının kullanım kolaylığına uygun olarak, ses üretim mekanizması ise icracının fiziksel limitleri göz ardı edilerek tasarlanabilmektedir (Mann, 2007). Fakat akustik müzik enstrümanlarının müzikal kontrol mekanizmaları ile ses üretim mekanizmaları fiziksel sınırlılıklar dolayısıyla birbirlerine bağlıdır.

Elektronik müzik enstrümanlarının gelişimi ile birlikte müzikal kontrolcüler ses üretim mekanizmaları olmayan, icracının hareketlerini farklı şekillerde yakalayıp ses üretim mekanizmasına iletebilen bağımsız cihazlar olarak tasarlanmaya başlamışlardır. Bu nedenle akustik müzik enstrümanlarının aksine, elektronik müzik enstrümanlarının kontrol mekanizmaları ile ses üretim mekanizmaları çoğunlukla birbirine fiziksel olarak bağlı değildir. İcracı, kullandığı elektronik müzik enstrümanın ses üretim mekanizmasının kabul ettiği standartta sinyaller üretebilen herhangi bir müzikal kontrolcüyü kullanmakta serbesttir. Bu kontrolcüler arasında taşınabilir ve giyilebilir klavyeler, ribbon kontrolcüler, dokunma hassasiyetli butonlar ve sensörler aracılığıyla elde edilen beyin sinyallerini kullanan kontrolcüler de bulunmaktadır (Paradiso ve O'Modhrain, 2003). Dijital teknolojilerin hayatımıza girmesiyle birlikte müzikal kontrolcülerin de değiştiğini ve geliştiğini gözlemlemek mümkündür.

1970'lerin sonlarına doğru sentezleyiciler dijital elektronik devreler kullanılarak tasarlanmaya başlamıştır. Fakat bu dönemde sentezleyici üreticileri arasında bir iletişim standardı belirlenmemiş olmasından dolayı farklı markaların sentezleyicileri ve müzikal kontrolcüleri birbirleriyle uyumlu çalışmamaktadır. Sequential Circuits şirketi, bütün sentezleyiciler tarafından ortak kullanılabilecek bir dijital iletişim standardı öneren ilk şirket olmuştur. İlk olarak evrensel sentezleyici arabirimi (universal synthesizer interface, USI) olarak isimlendirilen bu iletişim standardı daha sonrasında MIDI olarak adlandırılmıştır (Lehrman ve Tully, 1993).

1980'lerin başlarında, MIDI'nin de ortaya çıkması ile birlikte, müzikal kontrolcü ile sentezleyici basit bir seri kablo ile birbirine bağlanabilir hale gelmiştir. Böylece bu cihazlar arasındaki ayrım daha da belirginleşmiştir (Paradiso ve O'Modhrain, 2003). MIDI ile birlikte müzikal kontrolcüler ile sentezleyiciler arasındaki iletişimin standartlaştırılması, MIDI teknolojisini destekleyen her müzikal kontrolcü ile sentezleyicinin birlikte kullanılabilmesine olanak sağlamıştır ve bu sayede farklı müzikal kontrolcülerin tasarımı kolaylaşmıştır (Jordà, 2020).

2.1.1. MIDI

MIDI; yazılımsal ve donanımsal müzikal enstrümanların, bilgisayarların, kontrolcülerin ve benzer özellikli diğer cihazların birbirleriyle haberleşmelerini sağlamak adına oluşturulmuş kontrol dili ve donanımsal özelliklerin tamamını kapsayan bir dijital haberleşme protokolü olarak tanımlanabilir (Huber, 2020).

MIDI iletişim protokolü, Sequential Circuits'in kurucusu Dave Smith ve Chet Wood tarafından 1981 yılında düzenlenen 70. AES kongresinde USI adıyla tanıtılmıştır (Anderton, 2013; Smith ve Wood, 1981). USI standardı 1982 yılında Japon ve Amerikan enstrüman şirketlerinin çalışmalarıyla geliştirilerek MIDI ismiyle halka tanıtılmıştır. MIDI standardını kullanan ilk enstrüman ise Aralık 1982 tarihinde Roland tarafından piyasaya sürülen Sequential Circuits Prophet-600 olmuştur. Ocak 1983 tarihinde düzenlenen NAMM etkinliği sırasında Sequential Circuits tarafından üretilen Prophet-600 ile Roland tarafından üretilen JP6 arasında MIDI bağlantısı kurulmuş ve protokol ilk defa halka açık bir gösteride denenmiştir (The MIDI Association, t.y.). MIDI protokolü uzun bir süre boyunca aynı kalsa da teknolojinin hızla gelişmesi MIDI protokolünün de gelişmesine sebep olmuştur.

Günümüzde MIDI kontrol dili iki farklı versiyondan oluşmaktadır. MIDI standardı ilk tanıtıldığında oluşturulmuş olan standart MIDI 1.0 olarak ve 2019 NAMM konferansında tanıtılan yeni standart MIDI 2.0 olarak adlandırılmaktadır. MIDI 2.0, MIDI 1.0 üzerinde yapılan geliştirmelerle birlikte önceki versiyonla uyumlu bir şekilde tasarlanmıştır (Huber, 2020). Bu sayede iki standardı da kullanan müzikal kontrolcülerin ve sentezleyicilerin uyumluluğu korunmuştur.

MIDI kontrol dili genel kullanım amacıyla kullanılabilecek bir iletişim protokolü olsa da müzikal bilginin iletilmesi için özelleştirilmiştir. Bu çalışmada kullanılacak cihazların tamamı MIDI 1.0 kontrol dilini kullanmaktadır.

2.1.1.1.MIDI 1.0 kontrol dilinin özellikleri

MIDI, durum baytı ve veri baytı olarak isimlendirilen 8-bit uzunluğundaki kelimeler (bayt) ile haberleşmektedir. MIDI durum baytı, mesajın gönderileceği cihazın yerine getireceği MIDI fonksiyonuna ve kanala dair bilgiyi taşır. MIDI veri baytı ise gönderilen durum mesajının belirlediği fonksiyona uygun verilerin gönderilmesi için kullanılmaktadır (Huber, 2020; MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996).

MIDI mesajları, kanal mesajları ve sistem mesajları olarak iki farklı kategoriye ayrılmaktadır. MIDI kanal mesajları bir durum baytı ile bir veya iki veri baytından oluşmaktadır. MIDI kanal mesajlarının durum baytı kanalın yerine getireceği işlevi ve kanal bilgisini taşımaktadır. MIDI kanal mesajları 16 farklı kanal üzerinden gönderilebilir. Dolayısıyla sadece tek bir MIDI kontrolcüsü kullanılarak 16 farklı ses sentezleyicisi kontrol edilebilir. MIDI kanal mesajları şunlardır (MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996):

a. Note-Off; basılı tutulan notaya basılmayı bırakıldığı zaman gönderilen mesajdır. Velocity ve nota numarası olmak üzere iki veri baytına sahiptir.
b. Note-On; yeni bir notaya basıldığı zaman gönderilen mesajdır. Velocity ve nota numarası olmak üzere iki veri baytına sahiptir.
c. Poly Key Pressure; basılan nota üzerindeki baskı şiddeti değiştiği zaman gönderilen mesajdır. Basınç bilgisi ve nota numarası olmak üzere iki veri baytına sahiptir.

d. Control Change; nota mesajı dışındaki anlık ya da sürekli kontrol mesajlarının gönderilmesi için kullanılan mesajdır. Kontrol numarası ve kontrol değeri olmak üzere iki veri baytına sahiptir.
e. Program Change; gönderildiği cihaz üzerinde kayıtlı sesler arasında geçiş yapmak için gönderilen mesajdır. Program numarasını niteleyen tek bir veri baytına sahiptir.
f. Channel Pressure; ilgili kanalda basılı olan bütün notalar üzerindeki baskı şiddetinin değiştirilmesi için kullanılan mesajdır. Basınç bilgisini niteleyen tek bir veri baytına sahiptir.
g. Pitch Bend; ilgili kanalda basılı olan bütün notaların perdesini yarım perde aralığında değiştirmek için gönderilen mesajdır. Değişim miktarı olarak tek değere sahip olsa da daha hassas veri barındırabilmesi için iki veri baytına sahiptir.

MIDI sistem mesajları ise ortak sistem mesajları, gerçek zamanlı sistem mesajları ve özel sistem mesajları olmak üzere üç farklı kategoride incelenebilir. Ortak kanal mesajları şunlardır (MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996):

a. MIDI Time Code Quarter Frame: MIDI iletişim protokolünü kullanan bir sistemin hareketli görüntü ile senkronizasyonunu sağlamak için kullanılan mesajdır. Hareketli görüntünün her karesinin çeyrek zamanında gönderilir ve sekiz mesajda – bir diğer ifadeyle iki kare süresinde – tamamlanır. İlk karenin başlangıcından itibaren kare, saniye, dakika ve saat bilgilerinin her biri ikişer mesajda iletilerek sekiz mesaj sonunda tam kare bilgisi aktarılmış olur. Gönderilecek zaman bilgisinin türünü ve zaman bilgisini içeren bir veri baytına sahiptir (MIDI Manufacturers Association Incorporated, t.y.).
b. Song Position Pointer: Parçanın başlangıcından itibaren kaç 16'lık notanın geçtiğini gösteren mesajdır. Pozisyon bilgisinin daha hassas bir mesaj olması dolayısıyla iki veri baytına sahiptir.
c. Song Select: Kayıtlı parçalar arasında geçiş yapmak için kullanılan mesajdır. Parça numarasını içeren bir veri baytına sahiptir.
d. Tune Request: Analog sentezleyicilerin osilatörlerinin akortlanması gerektiği bilgisini ileten mesajdır. Veri baytına sahip değildir.

e. End of Exclusive: Özel sistem mesajı gönderiminin bittiğini belirten mesajdır. Veri baytına sahip değildir.

Gerçek zamanlı sistem ise mesajları şunlardır (MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996):

a. Timing Clock: Her dörtlük nota için 24 eşit zaman aralığında gönderilen zamanlama mesajıdır. Veri baytına sahip değildir.
b. Start: Parçanın başlatılması gerektiğini belirten mesajdır. Veri baytına sahip değildir.
c. Continue: Parça durduruldu ise kaldığı yerden devam etmesi gerektiğini belirten mesajdır. Veri baytına sahip değildir.
d. Stop: Parçanın durdurulması gerektiğini belirten mesajdır. Veri baytına sahip değildir.

Gerçek zamanlı sistem mesajları iki sistem arasındaki senkronizasyonu sağladığı için diğer mesajlardan daha önceliklidir. Bu sebeple gerçek zamanlı sistem mesajları gerektiğinde diğer mesajların arasında da gönderilebilir (MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996).

Özel sistem mesajları, tanımlanan standart MIDI mesajlarına uymayan, üretici tarafından belirlenmiş özel mesajların iletilmesi için kullanılmaktadır. Özel sistem mesajlarının alıcı sistemin sınırlılıkları dışında herhangi bir veri baytı kısıtlaması yoktur. Mesajın boyutu değişken olabileceği için özel sistem mesajları her zaman "End of Exclusive" mesajı ile bitmek zorundadır. Özel sistem mesajları kategorisinde gerçek zamanlı mesajların iletilmesi için Real Time System Exclusive ve gerçek zamanlı olmayan mesajların iletilmesi için Non-Real Time System Exclusive olmak üzere iki farklı mesaj tanımlanmıştır (MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee, 1996).

Günümüzde dijital müzikal kontrolcülerin çoğunluğunun MIDI iletişim protokolünü kullandığı bilinmektedir. Bu nedenle MIDI iletişim protokolünü kullanan müzikal kontrolcüler genellikle MIDI kontrolcüleri olarak adlandırılmaktadır.

2.1.2. MIDI Kontrolcüleri

MIDI kontrolcüleri MIDI iletişim protokolünü kullanarak başka yazılım ve/veya donanımları kontrol etmek için tasarlanmış ve kendi ses üretim mekanizmaları

bulunmayan cihazlar olarak tanımlanabilir. Bu yapısı nedeniyle MIDI kontrolcüler hem müzik üretimi için hem de müzik üretimi dışındaki ışık kontrolü, görüntü kontrolü gibi işlemler için de uygundur (Huber, 2020).

Ses üretimi ve tasarımı ile ilgili işlemler için kullanılan MIDI kontrolcüleri üzerindeki kontroller, müzik performansı ögeleri ve kontrol ögeleri olmak üzere ikiye ayrılabilir. Müzik performansı ögeleri daha çok çalgı biçimindedir. Kontrol ögeleri ise ses seviyesi, panaroma, sinyal işlemcisi kontrolü gibi işlemler için kullanılmaktadır.

2.1.2.1.MIDI kontrolcüleri üzerindeki müzik performansı ögeleri

MIDI teknolojisinin en yaygın kullanım şekillerinden birisinin müzik performansı olduğu açıktır. MIDI kontrolcülerinde müzik performansı için kullanılan ögelerden bazıları şunlardır (Huber, 2020):

a. Klavye: Piyano klavyesine benzer yapıda olan bu ögeler her bir nota için tek bir MIDI mesajı göndermektedir. Bu kontrolcülerden bir kısmı sadece notaya basıldığı/çekildiği bilgisini gönderirken bir kısmı ise basma şiddeti bilgisini de gönderebilmektedir. Bu ve benzeri bilgilerin gönderilmesi için klavye tuşları kullanılmaktadır. MIDI kontrolcülerinde en yaygın kullanılan müzik performansı ögesidir. Klavye ögelerinin bulunduğu kontrolcülere örnek olarak Oxygen Pro 61 (M-Audio, t.y.) ve Komplete Kontrol S61 MK2 (Native Instruments GmbH, t.y.) gösterilebilir.
b. Sürekli klavyeler: Klavye ögelerine benzer olan bu ögeler üzerindeki tuşlar yukarı-aşağı ve/veya notalar arası sürekli geçişe izin vermektedir. Bu sayede, örneğin, nota üzerinde yukarı-aşağı hareket ses üreticisi tarafından üretilen sesin yüksekliğine, notalar arası hareket perde bilgisinin değiştirilmesini sağlayabilmektedir. Sürekli klavye ögelerinin bulunduğu kontrolcülere örnek olarak Continuum (Haken, Tellman ve Wolfe, 1998) ve Seaboard (Vincent, 2015) gösterilebilir.
c. Davul padleri: Çoğunlukla davul seslerinin çalınması için kullanılan bu ögeler basınca duyarlı butonlardan oluşmaktadır ve davul bagetleri ile çalınmaya uygun olarak üretilirler. Davul padlerinin bulunduğu kontrolcülere DrumKAT (Alternate Mode Inc., t.y.) ve DM10 MKII Pro Kit (inMusicBrands LLC, t.y.) örnek olarak gösterilebilir.

d. Adım sıralayıcılar: Davul seslerinin veya melodilerin ardışık olarak çalınması için programlanabilen müzikal kontrol ögeleridir. Adım sıralayıcılar 16, 32 veya 64 gibi adım sınırlarına sahip olabilir. Adım sıralayıcıların bağlı oldukları MIDI cihazlarla eşzamanlı çalışabilmeleri için MIDI timing clock bilgisini üretmeleri ya da bağlı bulundukları cihazdan almaları gerekmektedir. Adım sıralayıcıların bulunduğu kontrolcülere BeatStep Pro (Perrier, t.y.) ve SQ-64 (KORG Inc., t.y.) örnek olarak gösterilebilir.

Müzik performansı odaklı olarak üretilen bu cihazlar MIDI enstrümanları olarak isimlendirilmektedir. MIDI enstrümanları müzik performansı için üretilmiş olsalar dahi üzerlerinde kontrol ögelerini de bulundurmaktadır. Bu kontrol ögeleri hem ses tasarımı hem de ses kayıt sürecinde ihtiyaç duyulabilecek kontrollerdir (Huber, 2020).

2.1.2.2.MIDI kontrolcüleri üzerindeki kontrol ögeleri

Ses tasarımı ve ses kayıt sürecinde müzik performansı ögelerinin yanı sıra kontrol ögeleri de oldukça önemli bir yere sahiptir. MIDI enstrümanlarında parametrelerin kolaylıkla değiştirilebilmesini sağlayan kontrol ögeleri, DAW'ların, harici MIDI cihazları ile kontrol edilebilmesini de sağlayabilmektedir. Dijital ses atölyelerinin MIDI ile kontrol edilebilmesini sağlayan harici MIDI cihazları DAW kontrolcüleri olarak isimlendirilmektedir.

DAW kontrolcüler hem analog ses mikserlerinin üzerinde bulunan kontrollerin DAW ile birlikte kullanılabilmesini, hem de bu yazılımlar üzerindeki dijital ses işlemcilerin kontrol edilmesini sağlamaktadır. Dijital ses istasyonlarının kontrol edilmesi için üretilen kontrolcülerde yaygın olarak bulunan MIDI kontrol ögelerine sürgülü ya da döner direnç (fader), döner kodlayıcı (rotary encoder) ve butonlar örnek olarak verilebilir. Bu kontrol ögelerinin bir kısmı ya da tamamı müzik performansı ögelerine sahip olan MIDI kontrolcülerinde de bulunabilmektedir (Huber, 2020). X-TOUCH (Music Tribe, t.y.), FaderPort 16 (PreSonus Audio Electronics Inc., t.y.) ve Mackie Control Universal (Loud Audio LLC, t.y.) DAW kontrolcülerine örnektir.

MIDI kontrolcüleri üzerinde bulunan bir diğer yaygın kontrol ögesi ise ızgaradır. Bu kontrol ögesi hem müzik performansı hem de kontrol için kullanılabilmektedir (Rossmy ve Wiethoff, 2021).

Bu çalışmada ızgara kontrol ögesine sahip müzikal kontrolcüler müzikal ızgara kontrolcüler olarak anılmıştır.

2.1.3. Müzikal Izgara Kontrolcüler

Izgara kontrolcüler her bir hücrenin giriş öğesi olarak da kullanılabildiği düşük çözünürlüklü ekranlar olarak tanımlanabilir. Izgara kontrolcülerin müzikal ızgara kontrolcü olarak değerlendirilebilmesi için aşağıdaki özelliklere sahip olmaları gerektiği belirtilmiştir (Rossmy ve Wiethoff, 2021):

1. Görünür boyutta, ortogonal biçimde dizilmiş, en az 2 sıra ve 2 sütundan oluşan hücrelere sahip olmalıdır.
2. Hücreler fiziksel olarak kontrolcü yüzeyinden ayırt edilebilir olmalıdır.
3. Hücreler hem giriş hem de çıkış için kullanılabilmelidir fakat etkileşim ve geri bildirim birbirinden bağımsız olmalıdır.

Yukarıdaki gereklilikler göz önünde bulundurulduğunda, yüksek çözünürlüklü dokunmatik ekranlar (akıllı telefon, tablet, vb.), adım sıralayıcılarda yaygın olarak kullanılan tek sıra halindeki görsel geri bildirimli butonlar ve görsel geri bildirime sahip olmayan davul kontrolcüleri ızgara kontrolcü olarak değerlendirilmemektedir (Rossmy ve Wiethoff, 2021).

Izgara kontrolcüler, görsel geri bildirim sistemlerinin düşük çözünürlüklü ekranları andırması dolayısıyla geleneksel raster ekranlar ile benzerlik göstermektedir. Dolayısıyla ızgara kontrolcülerin bir yazılım desteği vasıtasıyla grafiksel kullanıcı arayüzü oluşturmak için kullanılması mümkündür.

Müzikal ızgara kontrolcüler üzerinde GUI oluşturmak için bir diğer seçenek yazılımsal GUI çatılarının kullanılmasıdır. Bilgisayarlar ile birlikte kullanılan raster ekranlar için geliştirilmiş birçok GUI çatısı mevcuttur ve bu çatılar raster ekranın tipinden, çözünürlüğünden ve bilgisayar ile iletişim yönteminden bağımsız olarak çalışabilmektedirler. Ancak raster ekranların çok yüksek çözünürlüklü olması sebebiyle bu ekranlar için geliştirilmiş GUI çatılarının ızgara kontrolcüler ile kullanılması ve/veya adapte edilmesi oldukça güçtür (Hoggenmueller, Tomitsch ve Wiethoff, 2018). Aynı zamanda farklı müzikal ızgara kontrolcüleri arasında kullanıcı girişlerinin iletilmesinde ve renk değerlerinin alınmasında bir standart bulunmamaktadır. Bu sebeple müzikal ızgara kontrolcüler üzerinde GUI oluşturulması ve farklı kontrolcülerin ilettiği kullanıcı girişlerinin değerlendirilebilmesi bu cihazlar için geliştirilmiş bir GUI çatısı kullanılması mümkündür.

2.2. GUI Çatısı

Günümüzde akıllı telefonlar, bilgisayarlar ve benzeri araçların neredeyse tamamının insan-makine etkileşimi için GUI'lerden oluştuğunu gözlemlemek mümkündür. GUI yoluyla insan ile makine arasındaki etkileşim pencereler, menüler, butonlar ve ikonlar gibi grafiksel araçlarla sağlanmaktadır. Bu sayede çalışan programın detaylarının kullanıcıdan gizlenerek bu etkileşimin daha kolay olması sağlanmaktadır (Martinez, 2011).

Bilgisayarlar için GUI ilk olarak Dougles Engelbart tarafından 1968'de halka açık bir sunumda gösterilmiştir. GUI'ler ile etkileşimin en önemli araçlarından biri olan bilgisayar faresi de ilk defa bu sunumda tanıtılmıştır. "Bütün demoların atası (mother of all demos)" olarak adlandırılan bu sunum grafiksel kullanıcı arayüzlerinin gelişiminin başlangıcı olarak görülmektedir (Reimer, 2005).

GUI tasarımında çoğunlukla aynı grafiksel araçlar bir pencere üzerinde birden çok kere gösterilmektedir. Örneğin bir bilgisayar programında -farklı ikonlara ya da açıklamalara sahip olsalar da- birden çok buton bulunduğunu gözlemlemek mümkündür. Fakat benzer grafiksel araçların hem aynı programda hem de farklı programlar içerisinde tekrar geliştirilmesi yazılım geliştirici açısından zorlu bir süreçtir. Bu sebeple GUI oluşturmak için GUI çatısı kullanmak yazılım geliştiriciler arasında yaygın bir pratiktir (Kaisler, 2005).

Yazılım çatısı farklı uygulamalar içerisinde tekrar kullanılabilecek birden çok yazılımsal parçanın birlikte çalışarak bir problemi çözmesi üzerine tasarlanmış yapılar olarak tanımlanabilir. Bu yapının özelliklerinin, yapının kendisi değiştirilmeden, genişletilebilmesi yazılım çatılarının önemli bir özelliğidir. Yazılım çatısının yapısının genişletilmesini kolaylaştırması sebebiyle .çatı tasarımında genellikle nesne yönelimli programlama (object oriented programming, OOP) dilleri kullanılmaktadır (Kaisler, 2005).

Nesne yönelimli programlama konsepti ilk olarak 1962 yılında SIMULA programlama dili ile birlikte ortaya çıkmıştır ("Simula - Guide", 2021). Nesne yönelimli programlama dillerinin temel özellikleri kısaca şunlardır (Weisfeld, 2013):

- Kapsülleme (encapsulation): Veri (data) ve bu veri üzerinde işlem gerçekleştirecek program (code), nesne (object) adı verilen kapsüllerin içerisinde bulunmaktadır. Bu nesneler sınıf (class) adı verilen modeller

kullanılarak oluşturulurlar. Her sınıf, kendisinden oluşturulan nesnenin sahip olacağı verileri ve bu veriler üzerinde işlem yapacak programı tanımlayarak aynı özelliklere sahip birden çok nesnenin tek bir model kullanılarak oluşturulabilmesini sağlamaktadır. Bu özelliğe kapsülleme adı verilmektedir.

- Kalıtsallık (inheritance): Veri ve verinin işlenme yöntemini belirten bir sınıfın başka sınıflara kendi özelliklerini aktararak yeni sınıflar türetilmesini sağlamasına kalıtsallık adı verilmektedir. Temel sınıflara süper sınıf (superclass), bu sınıflardan türetilmiş sınıflara alt sınıf (subclass) adı da verilmektedir. Bu sınıflardan oluşturulan nesneler hem kalıtsallık ile aldıkları özelliklere hem de kendi sınıflarının tanımladıkları özelliklere sahip olurlar.
- Polimorfizm (polymorphism): Kalıtsallık ile türetilmiş alt sınıfların süper sınıflardan aldığı özellikleri değiştirmesine polimorfizm adı verilmektedir. Bu sayede genel özellikler süper sınıflar tarafından tanımlanırken türetilen alt sınıflar daha özel işlemleri gerçekleştirebilmektedir.

GUI çatısı tasarımında nesne yönelimli programlama dillerinin bu özellikleri yaygın olarak kullanılmaktadır.

3. YÖNTEM

Müzikal ızgara kontrolcüler standart tasarımlarında yukarıda bahsedilen özelliklere sahiptir. Ancak kimi yazılımsal işlemler ile yeni özellikler eklemek mümkündür.

Bu çalışma kapsamında farklı kontrolcüler üzerinde GUI araçlarının oluşturulması, kullanıcı girişlerinin alınması ve bu girişlerin işlenerek farklı fonksiyonların yerine getirilmesi gibi özelliklerin müzikal ızgara kontrolcülere eklenmesi planlanmaktadır. Fakat bu özelliklerin eklenmesi için ne gibi gereklilikler olduğunun tespit edilmesi gerekmektedir. Bu sebeple çalışmanın hangi yöntemle yapılacağının tespiti için önce ihtiyaçların belirlenmesi gereği hasıl olmuştur.

3.1. İhtiyaçların Belirlenmesi

Geliştiricilerin farklı üreticiler tarafından tasarlanmış müzikal ızgara kontrolcülere grafiksel kullanıcı arayüzü hazırlamasını kolaylaştırmak ve geliştirilen arayüzün benzer özellikteki diğer müzikal ızgara kontrolcülerle çalışmasını sağlamak için bir yazılım çatısı geliştirilmesi gerekmektedir. Bu çalışmada bu eksikliği tamamlamak için tezin en önemli unsurlarından biri olan PyxelWidgets çatısı geliştirilmiştir. PyxelWidgets çatısı, geliştirilme amacını yerine getirebilmesi için şu özelliklere sahip olmalıdır:

- Farklı iletişim protokollerini ve/veya yöntemlerini kullanarak yazılımın çalışacağı bilgisayar ile haberleşecek müzikal ızgara kontrolcülerle iletişim kurabilmelidir.
- Çatı tarafından desteklenmeyen iletişim protokollerini ve/veya yöntemlerini kullanan müzikal ızgara kontrolcüleri desteklemek amacıyla çatıda yapılması gereken değişiklik miktarı minimum olmalıdır.
- Düşük çözünürlüklü ekranlar için özel olarak geliştirilmiş grafiksel araçlara sahip olmalıdır.
- Oluşturulan grafiksel araçlar müzikal ızgara kontrolcüye gönderilecek renk değerlerini hesaplayabilmelidir.
- İcracının müzikal ızgara kontrolcü ile etkileşimini takip ederek oluşturulan araçların durumlarını bu etkileşime göre değiştirebilmelidir.

- Müzikal ızgara kontrolcülerin kullanım amacının müzik performansı ve kontrol olması sebebiyle, geliştirilecek GUI araçlarının da bu amacı karşılaması gerekmektedir.
- Hedef kitlesinin yazılım geliştiriciler değil programlama ile ilgilenen müzisyenler olması sebebi ile kolay kullanılabilir olmalıdır.

3.2. PyxelWidgets Çatısı

PyxelWidgets çatısı Python programlama dili kullanılarak geliştirilmiştir. Nesne yönelimli programlama stili ile modüler bir çatı olarak tasarlanmıştır. PyxelWidgets, dört farklı ana modülden (module) oluşmaktadır. Bu modüller kısaca şu şekilde açıklanabilir:

- Controllers: Müzikal ızgara kontrolcüler ile iletişim kurmak için tasarlanmış ana modüldür.
- Widgets: Klavye, adım sıralayıcı, buton, fader gibi yaygın olarak kullanılan müzik performansı ve kontrol ögelerinin bulunduğu ana modüldür.
- Window: Widgets modülü ile Controllers modülü arasında iletişim kurması için geliştirilen ana modüldür.
- Utils: Controllers, Widgets ve Window modülleri tarafından ortak olarak kullanılan modüllerin bulunduğu ana modüldür.

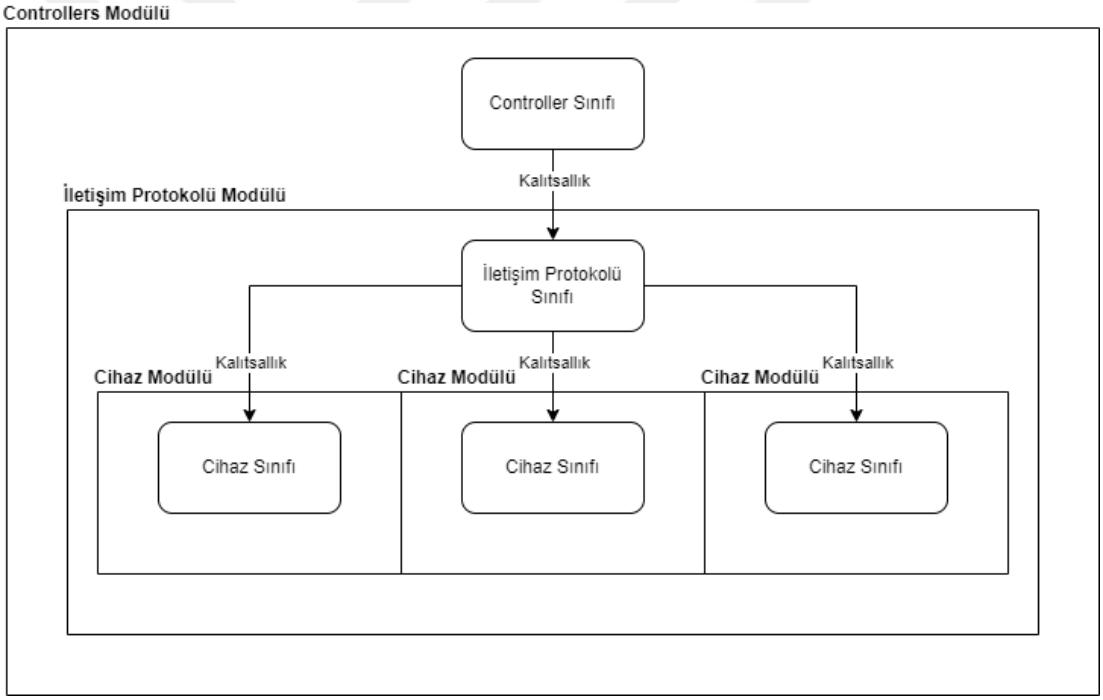
PyxelWidgets çatısında nesne yönelimli programlama dillerinin temel özellikleri olan kalıtsallık (inheritance) ve polimorfizm (polymorphism) yaygın olarak kullanılmıştır. Kalıtsallık özelliği kullanılarak yukarıda bahsedilen ana modüller içerilerinde ayrıca alt modüller geliştirilmiştir. Bu yolla elde edilen ana modül ve alt modüllerin farklı işlevleri vardır. Çatının ana modüllerindeki sınıflar daha genel işlevleri gerçekleştirirken bu sınıfların özelliklerini kalıtsallık yoluyla alan alt modüllerdeki sınıflar daha özel işlevleri gerçekleştirmektedir. Benzeri şekilde ana modüllerdeki sınıfların bazı fonksiyonları alt modüllerdeki sınıflar tarafından polimorfizm ile değiştirilerek aynı fonksiyonların farklı işlevleri yerine getirebilmesi sağlanmıştır. Bu sayede programcı, ihtiyacına yönelik olarak bu modüllerin bir kısmını ya da tamamını kullanmakta serbesttir.

PyxelWidgets çatısı Python programlama dilinin 3.10 versiyonu kullanılarak geliştirilmiş ve test edilmiştir. Bu çatı kullanılarak geliştirilen programların çalıştırılacağı bilgisayarda Python 3.10 yorumlayıcısının kurulu olması gerekmektedir.

PyxelWidgets çatısında dizi (array) işlemlerinin yapılması için NumPy (Harris ve diğerleri, 2020) ve MIDI cihazları ile iletişim için python-rtmidi (Arndt, 2022) kütüphaneleri kullanılmıştır.

3.2.1. Controllers modülü mimarisi

Controllers modülü hem müzikal ızgara kontrolcü üzerindeki kullanıcı girişlerini almak hem de araçlar tarafından hesaplanan renk değerlerini müzikal ızgara kontrolcüye iletmekle görevlidir. Controllers modülünün mimarisi Şekil 3.1'de gösterildiği gibidir.



Şekil 3.1 Controllers modülü mimarisi

Controllers modülü kendi içerisinde Controller sınıfını barındırmaktadır. Bu sınıf bir müzikal ızgara kontrolcü ile iletişimde ihtiyaç duyulabilecek şu üye fonksiyonları barındırmaktadır:

- init ve close fonksiyonları müzikal ızgara kontrolcü ile bağlantı kurmadan önce ve bağlantısını kestikten sonra yapılması gereken işlemleri içermektedir.

- connect ve disconnect fonksiyonları müzikal ızgara kontrolcü ile bağlantı kuracağı ve bağlantısını keseceği anda yapılması gereken işlemleri içermektedir.
- setCallback fonksiyonu müzikal ızgara kontrolcü tarafından iletilen kullanıcı girişlerinin geliştiriciye bildirilmesi için oluşturulacak olay (event) sırasında hangi fonksiyonun çağrılacağını belirlemektedir. Böylece geliştirici bu girişleri çatı ile oluşturulacak araçlara iletebilmekte ya da kendisi işleyebilmektedir.
- getButton ve setButton fonksiyonları müzikal ızgara kontrolcü tarafından iletilen kullanıcı girişlerinin saklanmasını ve işlenmesini sağlamak için kullanılmaktadır.
    - setButton fonksiyonu kendisine gelen kullanıcı girişi bilgisini setState fonksiyonuna aktarmakta; setState fonksiyonu ise eğer butona basıldı ise setPressed fonksiyonunu, eğer buton bırakıldı ise setReleased fonksiyonunu çağırmaktadır.
    - setPressed fonksiyonu buton belirli süre içerisinde bırakılmazsa setHeld fonksiyonunu çağıracak bir zamanlayıcı kurmakla ve bir olay oluşturarak setCallback ile belirlenen fonksiyonu çağırmakla yükümlüdür.
    - setReleased fonksiyonu ise buton belirlenen zamandan daha kısa süre içerisinde bırakıldı ise setHeld fonksiyonunun zamanlayıcısını iptal etmekle ve bir olay oluşturarak setCallback ile belirlenen fonksiyonu çağırmakla yükümlüdür.
    - setHeld fonksiyonu çağırıldı ise bir olay oluşturarak setCallback ile belirlenen fonksiyonu çağırmakla yükümlüdür.
- setCustom fonksiyonu iletişim kurulacak kontrolcünün ızgara dışında sahip olduğu diğer kontrollerin kullanıcıya bir olay oluşturularak iletilmesini sağlamakla yükümlüdür.
- sendPixel fonksiyonu kontrolcü üzerindeki hücrelerden herhangi biri için renk bilgisinin aktarılmasını sağlamakla yükümlüdür.
- updateOne, updateRow, updateColumn, updateArea ve update fonksiyonları kontrolcü yüzeyindeki hücrelerin renklerinin değiştirilmesini sağlamak için kullanılmaktadır. Bu fonksiyonlar rengi değiştirilmek istenen hücrenin kontrolcünün sınırları içerisinde olup olmadığını ve kontrolcünün hâlihazırda gösterdiği rengin değiştirilmek istenen renkle aynı olup olmadığını kontrol

ederek sadece değiştirilmesi gereken renk değerlerinin sendPixel fonksiyonuna gönderilmesini sağlamaktadır. Böylece cihazlar arasındaki bant genişliği korunmaktadır.

Controller sınıfı aynı zamanda şu üye değişkenleri barındırmaktadır:

- name değişkeni Controller sınıfından ya da alt sınıflarından oluşturulan objelere geliştirici tarafından verilmiş veya otomatik atanmış ismi tutmaktadır. name değişkeni setCallback ile belirlenen fonksiyon çağrılırken kullanılarak belirlenen fonksiyonunun hangi Controller objesi tarafından çağrıldığını bildirmek için kullanılmıştır.
- rect değişkeni kontrolcünün genişliğini, yüksekliğini ve yerleşimini tutan değişkendir. Utils modülünün Rectangle alt modülünün Rectangle2D sınıfından üretilmiş bir objedir. Bu değişken kontrolcü üzerindeki hücrelerde yapılacak renk değişikliklerinin kontrolcünün sınırları içerisinde kalıp kalmadığını test etmek için kullanılmaktadır.
- heldTime değişkeni kontrolcü üzerindeki butonlardan herhangi birine saniye cinsinden ne kadar süre basılı tutulduğunda setHeld fonksiyonunun çağrılacağını belirlemek için kullanılmaktadır.
- initialized, connected ve terminated değişkenleri kontrolcü ile yapılan bağlantının durumunu takip etmek için kullanılmaktadır.
- buffer değişkeni kontrolcü üzerinde o an gösterilen renk değerlerini tutan ve NumPy kütüphanesinin ndarray sınıfından üretilmiş bir objedir. Bu değişken kontrolcü üzerindeki hücrelerde yapılacak renk değişikliklerini takip etmek için kullanılmaktadır.
- buttons değişkeni kontrolcü üzerindeki butonların durumlarını tutan ve NumPy kütüphanesinin ndarray sınıfından üretilmiş bir objedir. Bu değişken kontrolcü üzerindeki kullanıcı giriş değişikliklerini takip etmek için kullanılmaktadır.
- clock değişkeni senkronizasyon sinyaline ihtiyaç duyan araçların kullanması için Utils modülünün Clock alt modülünün Clock sınıfından üretilmiş bir objedir. Her Controller objesi bir Clock objesine sahip olsa da bu obje kendi zamanlama sinyalini üretmek yerine başka zamanlama sinyali üreten kaynaklar ile senkronize edilebilir.
- callback değişkeni setCallback ile belirlenen fonksiyonun tutulması için kullanılmaktadır. Bu değişkene çağrılabilir (callable) bir değer atanmalıdır ve

Controller sınıfı tarafından çağrılırken kullanılacak argümanları kabul etmelidir.

Controller sınıfı, Controllers modülünün alt modülleri tarafından kalıtsallık yoluyla daha özel sınıfların üretilmesi için geliştirilmiştir ve tek başına bir işleve sahip değildir. Fakat bütün alt sınıflar tarafından kullanılabilecek genel işlevleri tanımlaması sayesinde bu işlevlerin her alt sınıf tarafından tekrar tanımlanması ihtiyacını ortadan kaldırmaktadır.

PyxelWidgets çatısı müzikal ızgara kontrolcüler MIDI mesajları yoluyla iletişim kurmak için MIDI iletişim protokolünü desteklemektedir. Bu sebeple Controllers modülüne MIDI alt modülü tasarlanmıştır.

3.2.1.1.MIDI alt modülü mimarisi

MIDI alt modülü MIDI iletişim protokolünü kullanan kontrolcüler için tasarlanan ana modüldür. MIDI modülü içerisinde Controller sınıfının kalıtsal özelliklerine sahip bir MIDI sınıfı geliştirilmiştir. Bu sınıf MIDI iletişim protokolünü kullanan kontrolcüler için tasarlanan ana sınıftır.

MIDI sınıfı, Controller sınıfından gelen fonksiyonların dışında şu üye fonksiyonlara sahiptir:

- listInputDevices, listOutputDevices ve listIODevices statik fonksiyonları yazılımın çalıştığı bilgisayara bağlı bulunan MIDI cihazlarının isimlerinin konsola yazdırılmasını sağlamaktadır. Bu sayede geliştirici bağlanmak istediği MIDI cihazının ismine erişilebilmektedir.
- connectVirtual fonksiyonu yazılımın çalıştığı bilgisayarda sanal bir MIDI bağlantı noktası oluşturularak bu MIDI bağlantı noktasına bağlanmak için kullanılmaktadır. Bu sayede geliştirici PyxelWidgets çatısı tarafından oluşturulan araçlarda kullanıcı tarafından yapılan değişiklikleri MIDI protokolünü destekleyen diğer yazılımlara (dijital ses istasyonu, yazılımsal sentezleyici vb.) aktarabilmektedir.
- processMIDI fonksiyonu bağlanılan MIDI cihazından gelen girişlerin işlenmesi için kullanılmaktadır.
- sendSysex, sendNoteOff, sendNoteOn, sendAftertouch, sendControlChange, sendProgramChange, sendChannelAftertouch, sendPitchBend ve

sendMessage fonksiyonları bağlanılan MIDI cihazına standart MIDI mesajlarının gönderilebilmesini sağlamaktadır.

MIDI sınıfı, Controller sınıfından kalıtsal olarak aldığı şu üye fonksiyonları polimorfizm yoluyla değiştirmektedir:

- connect fonksiyonu MIDI cihazlarına bağlanmak için gerekli olan işlevleri yerine getirecek şekilde değiştirilmiştir. Bu fonksiyon ile bir MIDI cihazına bağlanmak için MIDI cihazının ismi verilmelidir.
- disconnect fonksiyonu MIDI cihazı ile yazılımın bağlantısını kesmek için gerekli olan işlevleri yerine getirecek şekilde değiştirilmiştir.

MIDI sınıfı Controller sınıfından gelen değişkenlerin dışında şu üye değişkenlere sahiptir:

- _midiInput değişkeni donanımsal MIDI cihazından gelen mesajların alınması için rtmidi modülünün MidiIn sınıfından üretilmiş bir objedir.
- _midiOutput değişkeni donanımsal MIDI cihazına mesaj gönderilebilmesi için rtmidi modülünün MidiOut sınıfından üretilmiş bir objedir.
- virtual değişkeni iletişim kurulacak MIDI bağlantı noktasının sanal bir cihaz olup olmadığını göstermektedir. MIDI cihazına connectVirtual fonksiyonu ile bağlanıldıysa bu değişken her zaman doğru (true) değerini tutmaktadır.

MIDI sınıfı, MIDI modülünün alt modülleri tarafından kalıtsallık yoluyla daha özel sınıfların üretilmesi için geliştirilmiştir. Bütün alt sınıflar tarafından kullanılabilecek genel işlevleri tanımlaması sayesinde bu işlevlerin her alt sınıf tarafından tekrar tanımlanması ihtiyacını ortadan kaldırmaktadır.

PyxelWidgets çatısının MIDI alt modüllerinde 13 farklı sınıf bulunmaktadır. Bu sınıflar farklı kontrolcüler ile iletişim için kullanılacak özel fonksiyonlara sahiptir. Bu sayede, farklı iletişim yöntemlerini kullanan cihazlar PyxelWidgets çatısı ile desteklenebilmektedir. Her bir sınıfın incelenmesi yerine farklı özelliklere sahip olmaları sebebiyle, Akai APC40 MK2 (inMusicBrands LLC, t.y.) kontrolcüsü ve Novation Launchpad MK3 (Novation Digital Music Systems Ltd, t.y.) serisi kontrolcüleri için geliştirilmiş sınıfların ne şekilde çalıştığı gözlemlenmiştir.

Akai APC40 alt modülünün mimarisi

MIDI alt modülü altındaki Akai APC40 alt modülünün sınıfı olan MK2 sınıfı Akai şirketi tarafından üretilen 2. nesil APC40 kontrolcüsü ile iletişim kurmak için geliştirilmiş ve MIDI sınıfının kalıtsal özelliklerine sahip bir sınıftır.

MK2 sınıfı, MIDI sınıfından gelen fonksiyonların dışında şu üye fonksiyonlara sahiptir:

- changeMode fonksiyonu cihazın üzerindeki ızgara hücrelerinin aldığı kullanıcı girişleri ile yansıttığı görsel geri bildirimin ayrılmasını sağlamak için kullanılmaktadır. APC40 MK2 cihazının Generic, Ableton ve Alternative olarak üç farklı modu bulunmaktadır. Bu cihazın PyxelWidgets çatısı ile birlikte kullanılabilmesi için Ableton moduna alınması gerekmektedir.

MK2 sınıfı MIDI sınıfından kalıtsal olarak aldığı şu üye fonksiyonları polimorfizm yoluyla değiştirmektedir:

- connect fonksiyonu bağlantı anında cihazın Ableton moduna alınmasını sağlayacak şekilde değiştirilmiştir.
- disconnect fonksiyonu bağlantı kesilme anında cihazın varsayılan modu olan Generic moduna alınmasını sağlayacak şekilde değiştirilmiştir.
- sendPixel fonksiyonu cihaza gönderilecek renk bilgisinin bu cihaza uygun olarak gönderilmesini sağlayacak şekilde değiştirilmiştir. Akai APC40 MK2 cihazında kendi içerisindeki 128 renklik paletten bir renk seçilebilmektedir. Utils modülünün Pixel alt modülünün Pixel sınıfının fonksiyonlarından olan findInPalette fonksiyonu ile palet içerisindeki renklerden hesaplanan renge en yakın renk bulunarak gösterilmesi sağlanmaktadır.
- processMIDI fonksiyonu, hem ızgara hücreleri hem de diğer kontroller üzerindeki değişiklikleri geliştiriciye bildirecek şekilde değiştirilmiştir.

Akai APC40 MK2 cihazı kullanıcı girişini hücre indisi ve velocity değerleri ile iletmektedir. Hücre indisi Note-On mesajının nota numarası ile, hücreye basıldığı bilgisi velocity baytının içerisindeki 127 değeri ile ifade edilmektedir. Velocity baytının 0 olması ise hücreye artık basılmadığını göstermektedir. Cihaza renk bilgisininin gönderilebilmesi için hücre ve renk indislerinin bildirilmesi gerekmektedir. Hücre indisi MIDI Note-On mesajının nota numarası ile, renk indisi ise velocity değeri ile iletilmektedir.

Novation Launchpad alt modülü mimarisi

MIDI alt modülü altındaki Novation Launchpad alt modülünün sınıflarından biri olan MK3 sınıfı, Novation şirketi tarafından üretilen Launchpad serisi kontrolcülerin 3. nesil kontrolcülerini kontrol etmek için geliştirilmiş ortak bir sınıftır. Bu nesildeki kontrolcülerin tamamı kullanıcı girişlerini ve renk bilgisini benzer MIDI mesajları kullanarak işlediği için tek bir sınıf tarafından desteklenmektedir. MK3 sınıfından obje üretimi sırasında bu nesildeki hangi cihazın kullanılacağının belirtilmesi gerekmektedir.

MK3 sınıfı MIDI sınıfından gelen fonksiyonlar dışında şu üye fonksiyonlara sahiptir:

- setLayout fonksiyonu ile cihaz üzerindeki farklı arayüzlerden biri seçilebilmektedir. Novation Launchpad MK3 cihazlarının PyxelWidgets çatısı ile birlikte kullanılabilmesi için Programmer arayüzünün seçilmesi gerekmektedir.
- setMode fonksiyonu farklı Novation Launchpad MK3 cihazlarında Programmer arayüzünün kolaylıkla seçilmesini sağlamaktadır.

MK3 sınıfı MIDI sınıfından kalıtsal olarak aldığı şu üye fonksiyonları polimorfizm yoluyla değiştirmektedir:

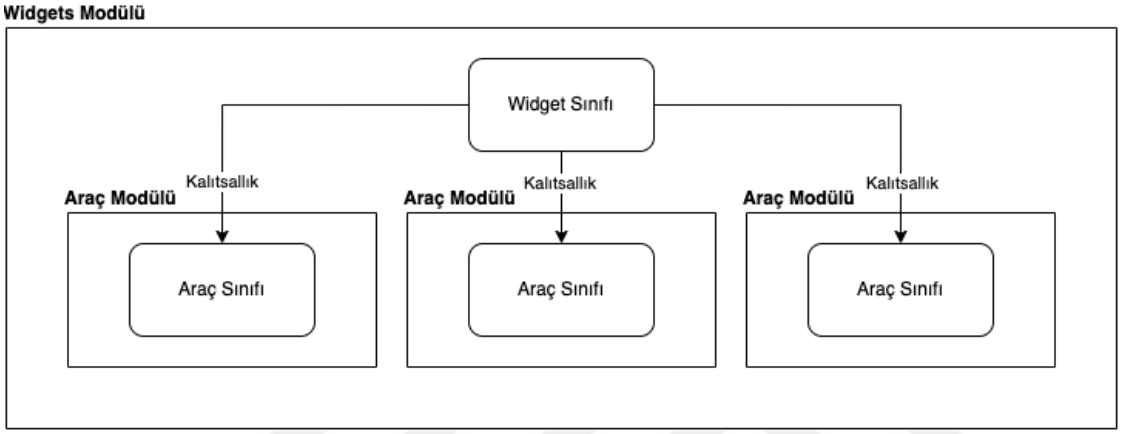
- connect fonksiyonu bağlantı anında cihazın Programmer arayüzüne alınmasını sağlayacak şekilde değiştirilmiştir.
- disconnect fonksiyonu bağlantı kesilme anında cihazın varsayılan arayüze alınmasını sağlayacak şekilde değiştirilmiştir.
- sendPixel fonksiyonu cihaza gönderilecek renk bilgisinin bir arabelleğe alınmasını sağlayacak şekilde değiştirilmiştir. Novation Launchpad MK3 nesili kontrolcüler birden çok hücre için renk bilgisinin tek MIDI Non Real-Time System Exclusive mesajı ile gönderilmesini kabul etmektedir.
- update, updateOne, updateRow, updateColumn ve updateArea fonksiyonları renk bilgisi gönderilmesi gereken bütün hücreler için sendPixel fonksiyonunu çağıran fonksiyonlardır. Bu fonksiyonlar, ilgili işlemi tamamladıktan sonra arabelleğe alınan renk bilgilerinin toplu olarak gönderilmesini sağlayacak şekilde değiştirilmiştir.

Yukarıdaki iki örnekte görüldüğü üzere Akai APC40 MK2 ve Novation Launchpad MK3 nesli cihazlar aynı iletişim protokolünü kullansalar dahi farklı MIDI mesajlarını kullanarak kullanıcı girişlerini iletmekte ve renk bilgisini almaktadır. PyxelWidgets

çatısının modüler yapısı sayesinde çatı üzerinde herhangi bir değişiklik yapmadan bu cihazların tamamının çatı ile birlikte çalışması sağlanabilmektedir.

3.2.2. Widgets modülü mimarisi

Widgets modülü müzikal ızgara kontrolcülere uygun olarak tasarlanmış GUI araçlarını içermektedir. GUI oluşturmayı sağlayan yazılım kütüphaneleri ve çatılarında bu araçlara widget adı verilmektedir (Kaisler, 2005). Bu sebeple bu modüle Widgets adı verilmiştir. Widgets modülünün mimarisi Şekil 3.2'de gösterildiği gibidir.



Şekil 3.2 Widgets modülü mimarisi

Widgets modülü kendi içerisinde Widget sınıfını barındırmaktadır. Bu sınıf kontrolcü için tasarlanacak araçlarda ihtiyaç duyulabilecek temel fonksiyonlardan, özelliklerden (property) ve değişkenlerden oluşmaktadır. Widget sınıfı şu üye fonksiyonları barındırmaktadır:

- setCallback fonksiyonu aracın sahip olduğu değerin değişmesi durumunda geliştiriciye haber verilebilmesi için hangi fonksiyonun çağrılacağını belirlemektedir. Bu sayede geliştirici bu değişiklikleri takip ederek aracın hangi işlevi gerçekleştireceğini belirleyebilmektedir.
- setValue fonksiyonu aracın değerini 0 ile 1 aralığında değiştirmek için kullanılmaktadır. Fonksiyon aracın değerinin değişmesi durumunda geliştiriciye haber vermektedir.
- process fonksiyonu kontrolcüden gelen kullanıcı girişi bilgisinin araç tarafından işlenmesi için kullanılan fonksiyondur. Bu fonksiyon gelen kullanıcı girişinin kendi sınırları içerisinde olup olmadığını kontrol ederek sadece kendi sınırları içerisindeki Pressed, Released ve Held olaylarına tepki vermektedir.

Bu olaylar process fonksiyonu tarafından Widget sınıfının fonksiyonlarından pressed, released ve held fonksiyonlarına iletilmektedir.

- pressed, released ve held fonksiyonları process fonksiyonu tarafından gelen olayları press, release ve hold fonksiyonlarına ilettikten sonra geliştiriciye haber vermekle yükümlüdür.
- press, release ve hold fonksiyonları pressed, released ve held fonksiyonları tarafından gelen olaylara karşı aracın vereceği tepkiyi belirlemektedir. Bu fonksiyonlar kullanıcı girişi olmaksızın aracın durumunun geliştirici tarafından değiştirilebilmesine izin vermektedir.
- tap ve tapped fonksiyonları fiziksel bir kullanıcı girişi olmasa dahi Widget objesindeki hücrelerden birine basılıp çekilme olayını taklit etmektedir. tap fonksiyonu press ve release fonsiyonlarını kullanırken tapped fonksiyonu pressed ve released fonksiyonlarını kullanmaktadır.
- update fonksiyonu kontrolcü tarafından gösterilmesi gereken renk bilgilerini aracın sınırlarına ve o anki durumuna göre hesaplamak için kullanılan fonksiyondur.
- _resize fonksiyonu oluşturulan aracın yeniden boyutlandırılması durumunda gerekli değişikliklerin yapılması için kullanılmaktadır.

Widget sınıfı aynı zamanda şu üye değişkenleri barındırmaktadır:

- name değişkeni Widget sınıfından ya da alt sınıflarından oluşturulan objelere geliştirici tarafından verilmiş veya otomatik atanmış ismi tutmaktadır. name değişkeni setCallback ile belirlenen fonksiyon çağrılırken kullanılarak, belirlenen fonksiyonunun hangi Widget objesi tarafından çağrıldığını bildirmek için kullanılmıştır.
- rect değişkeni Widget objesinin genişliğini, yüksekliğini ve yerleşimini tutan değişkendir. Utils modülünün Rectangle alt modülünün Rectangle2D sınıfından üretilmiş bir objedir. Bu değişken Widget objesine iletilen kullanıcı girişlerinin obje sınırları içerisinde olup olmadığının test edilmesi ve renk değerleri hesaplanırken hesaplanması gereken sınırların belirlenmesi için kullanılmaktadır.
- activeColor değişkeni aracın sahip olduğu 0 ile 1 arasındaki değere göre aktif olan hücrelerde gösterilmesi gereken rengi tutmaktadır.

- deactiveColor değişkeni aracın sahip olduğu 0 ile 1 arasındaki değere göre aktif olmayan hücrelerde gösterilmesi gereken rengi tutmaktadır.
- delta değişkeni aracın değerinin değiştirilmesi durumunda yeni değer ile eski değer arasındaki farkı tutmaktadır.
- bufferUpdated değişkeni araç tarafından hesaplanmış renk değerlerinin tekrar hesaplanması gerektiğini belirtmektedir. Renk değerlerinin tekrar hesaplanmasının gerekmediği durumlarda aracın işlemci üzerinde yük oluşturmasını engellemek için kullanılmaktadır.
- buffer değişkeni aracın hesapladığı renk değerlerini tutan ve NumPy kütüphanesinin ndarray sınıfından üretilmiş bir objedir. Bu değişken araç üzerindeki hücrelerde renk değişikliği yapılmasının gerekmediği durumlarda kullanılmaktadır.
- lock değişkeni Controller tarafından iletilen kullanıcı girişlerinin aracın değerini değiştirmesini engellemek için kullanılmaktadır. Aracın sadece görsel geri bildirim için kullanılmasını sağlamak için kullanılabilmektedir.
- _value değişkeni aracın 0 ile 1 arasındaki değerini tutmaktadır. Bu değişken renk değerlerinin hesaplanmasında kullanılan en önemli değişkendir.
- callback değişkeni setCallback ile belirlenen fonksiyonun tutulması için kullanılmaktadır. Bu değişkene çağrılabilir bir değer atanmalıdır ve Widget sınıfı tarafından çağrılırken kullanılacak argümanları kabul etmelidir.

Widget sınıfı aynı zamanda şu üye özellikleri barındırmaktadır:

- updated özelliği aracın update fonksiyonunda döndürdüğü renk değerlerinin tekrar hesaplanması gerektiğini belirtmektedir.
- width ve height özellikleri aracın genişliğinin ve yüksekliğinin değiştirilmesini sağlamak için kullanılmaktadır. Bu özelliklerin değeri değiştirildiğinde araç buffer değişkenini yeniden boyutlandırarak geliştiriciye haber vermektedir.
- value özelliği aracın 0 ile 1 arasındaki değerinin değiştirilmesini sağlamak için kullanılmaktadır. Bu özelliğin değeri değiştirildiğinde araç _value, delta ve bufferUpdated değişkenleri üzerinde değişiklik yapmaktadır. value özelliği kullanılarak aracın değeri değiştirildiğinde _value değişkeninin değerinin 0 ile 1 arasında olacağı garanti edilmektedir. Bu sayede oluşabilecek hataların ortaya çıkmasının önüne geçilmektedir.

PyxelWidgets çatısının Widgets modülünün 9 adet alt modülü bulunmaktadır. Her modül kendi ismine sahip bir sınıfı barındırmaktadır ve bu sınıflar Widget sınıfından kalıtsallıkla türetilmiştir. Bu sınıfların yapısı yerine işlevselliğinin daha önemli olduğu düşünülmektedir.

3.2.2.1.Button aracının özellikleri

Button sınıfı bir butonun basılması ya da bırakılması gibi yalnızca iki farklı bilgiye ihtiyaç duyulduğunda kullanılabilecek bir araçtır. Anlık (Button), anahtar (Switch) ve basılıp çekildiğinde anlık basılı tutulduğunda anahtar (Mixed) olmak üzere üç farklı moda sahiptir. Bu aracın kapladığı alanın tamamı tek bir buton olarak çalışmaktadır.

3.2.2.2.ButtonGroup aracının özellikleri

ButtonGroup sınıfı birden fazla butonun basılması ya da bırakılması gibi yalnızca iki farklı bilgiye ihtiyaç duyulduğunda kullanılabilecek bir araçtır. Bu aracın kapladığı alandaki her hücre farklı bir buton gibi çalışmakta fakat kullanıcı girişlerine ortak yanıt vermektedir. Her butonun ayrı değerlendirildiği anlık (Button), bir buton aktifleştirildiğinde diğer bütün butonların pasif hale geldiği anahtar (Switch) ve birden çok butonun aynı anda aktifleştirilebildiği çoklu (Multi) olmak üzere üç farklı moda sahiptir.

3.2.2.3.Fader aracının özellikleri

Fader aracı sürekli değerlerin değiştirilmesi için kullanılabilen, analog ses mikserlerinde kullanılan sürgülü dirençlere benzer şekilde tasarlanmış bir araçtır. PyxelWidgets çatısında bulunan en çok özelliğe sahip araçlardan biridir.

Fader aracı dikey (Vertical) ve yatay (Horizontal) olarak iki farklı yönde oluşturulabilir. Dikey olarak oluşturulmuş araç en alt noktayı minimum noktası, en üst noktayı maksimum noktası olarak kabul ederken yatay olarak oluşturulmuş araç en sol noktayı minimum noktası, en sağ noktayı maksimum noktası olarak kabul etmektedir. Eğer oluşturulan aracın ızgara (Grid) tipi basit (Simple) olarak seçildiyse, dikey yönlü aracın aynı yatay eksende sahip olduğu bütün hücreler aynı renk değerini gösterecek ve kullanıcı girişlerine aynı şekilde yanıt verecektir. Izgara tipi basit olarak seçilmiş ve yatay yönlü bir aracın ise aynı dikey eksendeki bütün hücreleri benzer şekilde davranacaktır.

Fader aracının ızgara tipi matris (Matrix) olarak seçilebilmektedir. Bu durumda yönü dikey olarak oluşturulmuş ve ızgara tipi matris olan bir aracın yatay eksendeki bütün hücreleri kullanıcı girişlerine farklı tepki verecek ve renk değerleri buna göre hesaplanacaktır. Sayısal olarak karşılaştırma yapılırsa 4 hücre yüksekliğe ve 4 hücre genişliğe sahip dikey yönlü, ızgara tipi basit bir araç toplamda 4 işlevsel hücreye sahipken ızgara tipi matris olan bir araç 16 işlevsel hücreye sahip olmaktadır.

Fader aracı 5 farklı tipte (Type) oluşturulabilmektedir. Aracın tipinin değiştirilmesi çoğunlukla görsel olarak etki göstermektedir. Bu tipler Single, BoostCut, Wrap, Spread ve Collapse olarak isimlendirilmektedir. Bu tipler kısaca şöyle açıklanabilir:

- Single tipi aracın değerine göre sadece belli bir sınırın içerisinde kalan hücreyi renklendirmektedir.
- BoostCut tipi, analog ses mikserlerinde pan kontrolü için kullanılan döner dirençlere benzetilebilir. Tam ortadaki hücreden ayrılan bu aracın değeri orta noktanın ilerisinde ise sadece ortadan yukarısı ya da sağı, orta noktanın gerisinde ise sadece orta noktadan aşağısı ya da solu renklendirilmektedir.
- Wrap tipi, analog ses mikserlerinde kullanılan sürgülü dirençlere benzetilebilir. Bu araç sadece aracın değerinin altındaki hücreleri renklendirmektedir.
- Spread tipi analog ekolayzer (equalizer) bant genişliği (bandwidth) kontrolüne benzetilebilir. Bu tipte, aracın değeri arttıkça hücrelerin renkleri ortadan yanlara doğru açılmaktadır.
- Collapse tipi Spread tipinin tersi olarak çalışır. Aracın değeri arttıkça hücrelerin renkleri yanlardan ortaya doğru kapanmaktadır.

Fader aracı 5 farklı modda (Mode) oluşturulabilmektedir. Aracın modunun değiştirilmesi çoğunlukla kullanıcı girişinin nasıl yorumlanacağını değiştirmektedir. Bu modlar Simple, Multi, Magnitude, Sensitive ve Relative olarak isimlendirilmektedir. Bu modlar kısaca şöyle açıklanabilir:

- Simple modu kullanıcı girişinin olduğu hücrenin minimum değerini aracın değeri olarak belirlemektedir.
- Multi modu kullanıcı girişinin olduğu hücrenin değer aralığını Fader sınıfının sahip olduğu resolution üye değişkenine bölerek her butonun daha küçük değişiklikler yapmasını sağlamaktadır. Kullanıcı son bastığı hücreye tekrar bastığında Fader aracının değeri daha küçük aralıklarla artacaktır.

- Magnitude modu kullanıcı girişinin olduğu hücrenin ağırlıklı değerini aracın değeri olarak belirlemektedir.
- Sensitive modu hücrelerinde basınç hassasiyeti bulunan ızgara kontrolcülerde kullanıcı girişi esnasındaki basınca göre aracın değerini belirlemektedir.
- Relative modu basılı tutulan hücrenin ağırlıklı değerini aracın değeri olarak belirleyerek aracın sınırları içerisindeki başka bir hücreye basılması durumunda basılı tutulan hücrenin sınırları içerisindeki değerlerden ikinci hücrenin ağırlıklı olduğu değeri aracın değeri olarak belirlemektedir. Bu mod multi modu ile magnitude modunun karmasıdır.

3.2.2.4.Keyboard aracının özellikleri

Keyboard aracı PyxelWidgets çatısının en temel müzik performansı ögesidir. Piyano klavyesine benzer şekilde tasarlanan aracın her hücresi bir MIDI nota numarasına karşılık gelmektedir.

Keyboard aracı için gam (Scale), bu gamın kök notası (Root) ve oktavı seçilebilmektedir. Araç 15 farklı gam desteklemektedir. Araç ile oluşturulan klavyenin kök notası, gam içerisindeki notalar ve gam dışı notalar farklı renklendirilmektedir.

Keyboard aracı 5 farklı tipte (Type) oluşturulabilmektedir. Bu tipler Keyboard, ChromaticVertical, ChromaticHorizontal, Diatonic, DiatonicVertical ve DiatonicHorizontal olarak isimlendirilmektedir. Bu tipler kısaca şöyle açıklanabilir:

- Keyboard tipi aracın piyano klavyesine benzer bir yapıda olmasını sağlamaktadır. Yüksekliği iki ya da ikinin katı olarak oluşturulmuş Keyboard aracının dikey ekseninde çift sıradaki hücreler piyano klavyesi üzerindeki beyaz tuşlara, tek sıradaki hücreler piyano klavyesi üzerindeki siyah tuşlara denk gelecek şekilde yerleştirilmektedir.
- DiatonicVertical tipi aracın sadece gam içerisindeki seslerden oluşan sıralı notaları sol alt hücreden başlayarak önce sağa ardından yukarı doğru sıralamasını sağlamaktadır. Keyboard sınıfının fold değişkeni ile bir üst sıradaki notanın bir alt sıradaki kaçıncı notadan başlayacağı belirlenebilmektedir.
- DiatonicHorizontal tipi nota değerlerini DiatonicVertical ile aynı şekilde hesaplamakta fakat notaların sol üst hücreden önce aşağı ardından sağa doğru sıralanmasını sağlamaktadır.

- Diatonic tipi DiatonicVertical tipine ek olarak fold değişkeninden etkilenmemektedir, bu sebeple notalar her zaman alt sıranın bittiği noktadan üst sıraya katlanmaktadır.
- ChromaticVertical tipi DiatonicVertical tipine ek olarak gam dışı sesleri de barındırmaktadır.
- ChromaticHorizontal tipi DiatonicHorizontal tipine ek olarak gam dışı sesleri de barındırmaktadır.

3.2.2.5.Knob aracının özellikleri

Knob aracı sürekli değerlerin değiştirilmesi için kullanılabilen, analog ses mikserlerinde kullanılan döner dirençlere benzer şekilde tasarlanmış bir araçtır.

Knob aracı oluşturulduğu alanın sadece dış çerçevesindeki hücreleri kullanmaktadır. Bu sebeple aracın sınırları içerisinde olsa dahi kullanılmayan hücrelere farklı bir araç yerleştirmek mümkündür. Fader aracı ile benzer şekilde Single, BoostCut, Wrap, Spread ve Collapse tiplerine sahiptir.

Knob aracının değeri, kullanıcının dış çerçevedeki hücreler ile etkileşime geçmesi ile değiştirilebilir. Bu hücreler tam ortadan ikiye ayrılmıştır. Orta noktanın sağındaki hücreler aracın değerini artırırken solunda kalan hücreler aracın değerini azaltmaktadır. Aracın değerinin sürekli artırılıp azaltılması kullanıcının ilgili hücreye ne kadar süre basılı tuttuğuna ve aracın update fonksiyonunun ne kadar hızlı çağrıldığına bağlıdır. Üst noktaya yakın hücreler aracın değerinin daha yavaş değişmesini sağlarken aşağıya doğru indikçe hücreler aracın değerinin değişmesine daha hızlı etki etmektedir.

3.2.2.6.Sequencer aracının özellikleri

Sequencer aracı bir müzik performansı ögesi olan adım sıralayıcıyı taklit etmek için geliştirilmiştir.

Sequencer aracının bir adım sınırı yoktur. Yerleştirildiği alan içerisindeki hücre sayısı adım sayısını karşılayacak boyutta değil ise gerekli olan sayfa sayısını hesaplayarak hücre sınırına geldiğinde bir sonraki sayfaya geçmektedir. Adımlar sıralayıcının kapladığı alanın sol altından önce sağa sonra yukarı doğru hareket etmektedir. Hücrelerin hangi süre değerine sahip olacağı belirlenebilmektedir.

Adımların sıralayıcıya yerleştirilmesi için kullanıcının ilgili hücreye basması yeterlidir. Adım sıralayıcı birden çok sayfaya sahip olabileceği için sayfalama sisteminin adımları takip edip etmeyeceği belirlenebilmektedir.

Sequencer aracının bir zamanlama sinyaline ihtiyacı vardır. Hesaplamanın doğru yapılabilmesi için bu zamanlama sinyalinin her 4'lük nota için kaç adet gönderileceği araca bildirilmelidir. Her yeni zamanlama sinyalinde aracın tick fonksiyonunun çağrılması gerekmektedir. Tick fonksiyonunun çağrılması için bir Clock objesi ya da dış kaynaktan gelen bir zamanlama sinyali kullanılabilir.

3.2.2.7.Sprite aracının özellikleri

Sprite aracı sabit ya da hareketli görsellerin ızgara kontrolcü üzerinde gösterilebilmesi için geliştirilmiştir. Kullanıcı girişlerine tepki vermemektedir fakat görünmez renk ile oluşturulmuş diğer araçların bu aracın altına yerleştirilmesi ile kullanıcı girişi alınabilmektedir.

3.2.2.8.Tracker aracının özellikleri

Tracker aracı adım sıralayıcıya benzer bir araçtır. Yerleştirildiği alan içerisindeki bütün sütunlar farklı bir adım sıralayıcı olarak değerlendirilebilir. Aracın boyutundan bağımsız olarak adım sayısı belirlenebilmektedir. Bu durumda araç o anki adıma göre yukarıdan aşağıya doğru kaymaktadır. Ayrıca aracın sayfa sayısı da belirlenebilmektedir. Bu sayede sayfa sınırına kadar her sayfa sonunda araç bir sonraki sayfaya geçecek ve ardından ilk sayfaya dönecektir.

Tracker aracı Sequencer aracında olduğu gibi bir zamanlama sinyaline ve zamanlama sinyalinin her 4'lük nota için kaç adet gönderileceği bilgisine ihtiyacı vardır.

3.2.2.9.XY aracının özellikleri

XY aracı 2 boyutlu bir Fader aracına benzetilebilir. Bu araç yatay ve dikey hücrelere bağlı olarak 2 farklı değere sahiptir.

XY aracının kullanıcı girişini yorumlaması Fader aracının Relative moduna benzemektedir. Basılı tutulan hücrenin ağırlıklı değerini aracın değeri olarak belirleyerek aracın sınırları içerisinde yatay veya dikey eksende başka bir hücreye basılması durumunda basılı tutulan hücrenin sınırları içerisindeki değerlerden ikinci hücrenin ağırlıklı olduğu değeri aracın değeri olarak belirlemektedir.

3.2.3. Window modülü mimarisi

Window modülü kontrolcüler (Controller sınıfı) ve GUI araçları (Widget sınıfı) arasında bağlantı kurmak için geliştirilmiştir. Bu modül Window ve Manager olmak üzere iki farklı sınıftan oluşmaktadır.

3.2.3.1.Window sınıfı mimarisi

Window sınıfı birden fazla aracın kontrolcü üzerinde gösterilmesini ve işlenebilmesini sağlayan sınıftır. Bu sınıf kontrolcüden aldığı kullanıcı girişlerini araçlara, araçlardan aldığı renk bilgilerini de birleştirerek kontrolcüye aktarmakla yükümlüdür.

Window sınıfı şu üye fonksiyonlara sahiptir:

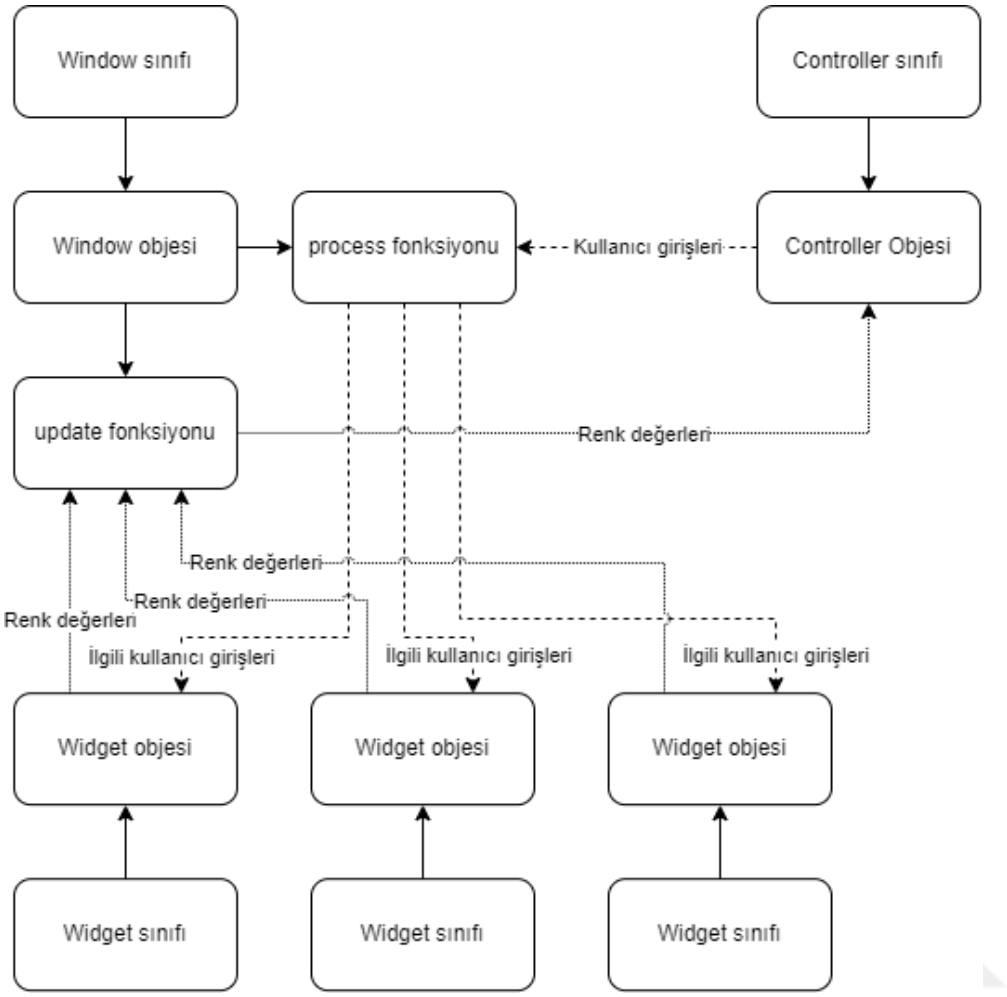
- addWidget fonksiyonu Window üzerine bir araç eklemek için kullanılmaktadır.
- addWidgets fonksiyonu Window üzerine birden fazla araç eklemek için kullanılmaktadır.
- removeWidget fonksiyonu Window üzerindeki araçlardan herhangi birinin silinmesi için kullanılmaktadır.
- forceUpdate fonksiyonu bütün araçların updated özelliğini doğru değerine eşitleyerek renk değerlerinin tekrar hesaplanmasını sağlamaktadır.
- process fonksiyonu kontrolcü tarafından iletilen kullanıcı girişlerinin araçlara aktarılmasını sağlamak için kullanılmaktadır.
- setCallback fonksiyonu process fonksiyonuna kontrolcü tarafından iletilen ızgara dışındaki kontrollerin değerlerindeki değişimin geliştiriciye haber verilebilmesi için hangi fonksiyonun çağrılacağını belirlemektedir.
- update ve updateArea fonksiyonları Window'un sınırları içerisindeki araçların update fonksiyonlarının çağrılmasını ve alınan renk değerlerinin birleştirilerek dışarıya aktarılmasını sağlamaktadır.

Window sınıfı aynı zamanda şu üye değişkenlere sahiptir:

- name değişkeni Window sınıfından ya da alt sınıflarından oluşturulan objelere geliştirici tarafından verilmiş veya otomatik atanmış ismi tutmaktadır. name değişkeni setCallback ile belirlenen fonksiyon çağrılırken kullanılarak belirlenen fonksiyonunun hangi Window objesi tarafından çağrıldığını bildirmek için kullanılmıştır.

- rect değişkeni oluşturulan Window'un genişliğini, yüksekliğini ve yerleşimini tutan değişkendir. Utils modülünün Rectangle alt modülünün Rectangle2D sınıfından üretilmiş bir objedir. Bu değişken Window objesine iletilen kullanıcı girişlerinin obje sınırları içerisinde olup olmadığının test edilmesi ve renk değerleri hesaplanırken hesaplanması gereken sınırların belirlenmesi için kullanılmaktadır.
- widgets değişkeni Window objesine eklenen her aracın ismiyle birlikte objesinin saklandığı isimli listedir (dictionary).
- buffer değişkeni araçlar tarafından hesaplanan renk değerlerinin tutulduğu NumPy kütüphanesinin ndarray sınıfından üretilmiş bir objedir. Bu değişken bütün renk değerlerinin kontrolcüye iletilmesi için kullanılmaktadır.
- frameCounter değişkeni Window objesinin üretiminden itibaren update fonksiyonunun kaç defa çağrıldığını tutmak için kullanılmaktadır.
- callback değişkeni setCallback ile belirlenen fonksiyonun tutulması için kullanılmaktadır. Bu değişkene çağrılabilir bir değer atanmalıdır ve Window sınıfı tarafından çağrılırken kullanılacak argümanları kabul etmelidir.

Bu sınıftan üretilen objelerin kullanım şekli Şekil 3.3'de gösterilmiştir.



Şekil 3.3 Window sınıfının kullanımı

3.2.3.2.Manager sınıfı mimarisi

Manager sınıfı birden çok Window'un birden çok kontrolcü üzerinde gösterilebilmesini sağlayan sınıftır.

Bu sınıf kontrolcülerden aldığı kullanıcı girişlerini ilgili Window objelerine, Window objelerinden aldığı renk bilgilerini de ilgili kontrolcülere aktarmakla yükümlüdür. Manager sınıfı şu üye fonksiyonlara sahiptir:

- addWindow fonksiyonu Window sınıfından üretilmiş bir objenin Manager objesine eklenmesini sağlamak için kullanılmaktadır. Window objesinin yerleştirileceği alan bu fonksiyon çağrılırken bildirilmelidir.
- removeWindow fonksiyonu Manager objesine eklenmiş Window objelerinden herhangi birinin silinmesini sağlamak için kullanılmaktadır.

- addController fonksiyonu Controller sınıfından üretilmiş bir objenin Manager objesine eklenmesini sağlamak için kullanılmaktadır. Controller objesinin yerleştirileceği koordinatlar bu fonksiyon çağrılırken bildirilmelidir.
- removeController fonksiyonu Manager objesine eklenmiş Controller objelerinden herhangi birinin silinmesini sağlamak için kullanılmaktadır.
- forceUpdate fonksiyonu Manager objesine eklenmiş bütün Window objelerinin sahip olduğu Widget objelerinin renk değerlerini tekrar hesaplamasını sağlamak için kullanılmaktadır.
- process fonksiyonu kontrolcüler tarafından iletilen kullanıcı girişlerinin ilgili Window objelerine iletilmesini sağlamak için kullanılmaktadır.
- setCallback fonksiyonu process fonksiyonuna kontrolcüler tarafından iletilen ızgara dışındaki kontrollerin değerlerindeki değişimin geliştiriciye haber verilebilmesi için hangi fonksiyonun çağrılacağını belirlemektedir.
- update fonksiyonu Manager objesine eklenmiş bütün Window objelerine bağlı araçların renk değerlerinin alınmasını ve ilgili kontrolcülere iletilmesini sağlamaktadır.

Manager sınıfı aynı zamanda şu üye değişkenlere sahiptir:

- name değişkeni Manager sınıfından ya da alt sınıflarından oluşturulan objelere geliştirici tarafından verilmiş veya otomatik atanmış ismi tutmaktadır. name değişkeni setCallback ile belirlenen fonksiyon çağrılırken kullanılarak belirlenen fonksiyonunun hangi Manager objesi tarafından çağrıldığını bildirmek için kullanılmıştır.
- rect değişkeni Manager objesinin genişliğini, yüksekliğini ve yerleşimini tutan değişkendir. Utils modülünün Rectangle alt modülünün Rectangle2D sınıfından üretilmiş bir objedir. Bu değişken Manager objesine eklenen Window ve Controller objelerinin bu objenin sınırları içerisinde olup olmadığını test etmek için kullanılmıştır.
- windows değişkeni Manager objesine eklenen her Window objesinin, isminin ve yerleştirileceği alanın birlikte saklandığı isimli listedir.
- controllers değişkeni Manager objesine eklenen her Controller objesinin, isminin ve yerleştirileceği alanın birlikte saklandığı isimli listedir.

- buffer değişkeni Window objeleri tarafından hesaplanan renk değerlerinin tutulduğu NumPy kütüphanesinin ndarray sınıfından üretilmiş bir objedir. Bu değişken renk değerlerinin kontrolcülere iletilmesi için kullanılmıştır.
- callback değişkeni setCallback ile belirlenen fonksiyonun tutulması için kullanılmaktadır. Bu değişkene çağrılabilir bir değer atanmalıdır ve Manager sınıfı tarafından çağrılırken kullanılacak argümanları kabul etmelidir.

3.2.4. Utils modülü mimarisi

Utils modülü PyxelWidgets çatısında bulunan diğer bütün modüller tarafından ortak olarak kullanılan modüllerin bulunduğu ana modüldür. Bu modüller kendi içlerinde birbirlerine bağımlı değillerdir ve bağımsız olarak kullanılabilmektedirler.

Utils modülü 7 adet alt modüle sahiptir. Bu modüller kısaca şu şekilde açıklanabilir:

- Clock modülü Clock ve Target olmak üzere iki farklı sınıfa sahiptir. Clock sınıfı belirli zaman aralıklarında kendisine verilen Target objelerinin çalışmasını sağlamaktadır. Hem Clock sınıfından hem de Target sınıfından oluşturulan objeler birbirlerinden tamamen bağımsız çalışabilmektedir. Böylece ana programın ya da birbirlerinin çalışma sürelerinden etkilenmemektedirler. Bu sınıflar genellikle zamanlama sinyalinin ihtiyaç duyulduğu durumlarda kullanılmaktadır.
- Enums modülü yalnızca Event sınıfına sahiptir. Event sınıfı Controller, Widget, Window ya da Manager tarafından oluşturulan olayların hangi türde olduğunu belirtmek için kullanılmaktadır.
- Parser modülü Parser adında tek bir sınıfa sahiptir. Bu sınıf JSON formatındaki serileştirilmiş bir metin dosyası aracılığıyla PyxelWidgets çatısının sahip olduğu sınıfların kullanılabilmesini sağlamaktadır.
- Pixel modülü Pixel adında tek bir sınıfa sahiptir. PyxelWidgets çatısında farklı sınıflar tarafından hesaplanan ya da tutulan bütün renk değerleri bu sınıftan oluşturulan objeler kullanılarak ifade edilmektedir. Bu sınıf renk bilgisini kırmızı, yeşil, mavi ve alfa kanalı kullanarak tutmaktadır.
- Rectangle modülü Position2D, Dimension2D ve Rectangle2D olmak üzere 3 farklı sınıftan oluşmaktadır. Position2D sınıfı 2 boyutlu bir alan içerisindeki bir objenin dikey ve yatay eksendeki koordinatlarını, Dimension2D sınıfı ise 2 boyutlu bir objenin yükseklik ve genişlik bilgilerini tutmaktadır. Rectangle2D

sınıfı bu iki sınıftan kalıtsallıkla türetilmiştir ve iki farklı Rectangle2D objesinin 2 boyutlu bir uzayda birbirleri ile çarpışma durumunu ve çarpışma boyutunu hesaplayabilmektedir.

- teVirtualMIDI modülü teVirtualMIDI adında tek bir sınıfa sahiptir. Bu sınıf PyxelWidgets çatısını kullanan programların Windows işletim sisteminde sanal MIDI bağlantı noktası oluşturabilmesini sağlamak için kullanılmaktadır. Bu bağlantı noktasının oluşturulabilmesi için programın çalıştığı bilgisayarda virtualMIDI (Erichsen, t.y.) sürücüsünün ve teVirtualMIDI dinamik kütüphanesinin yüklü olması gerekmektedir.

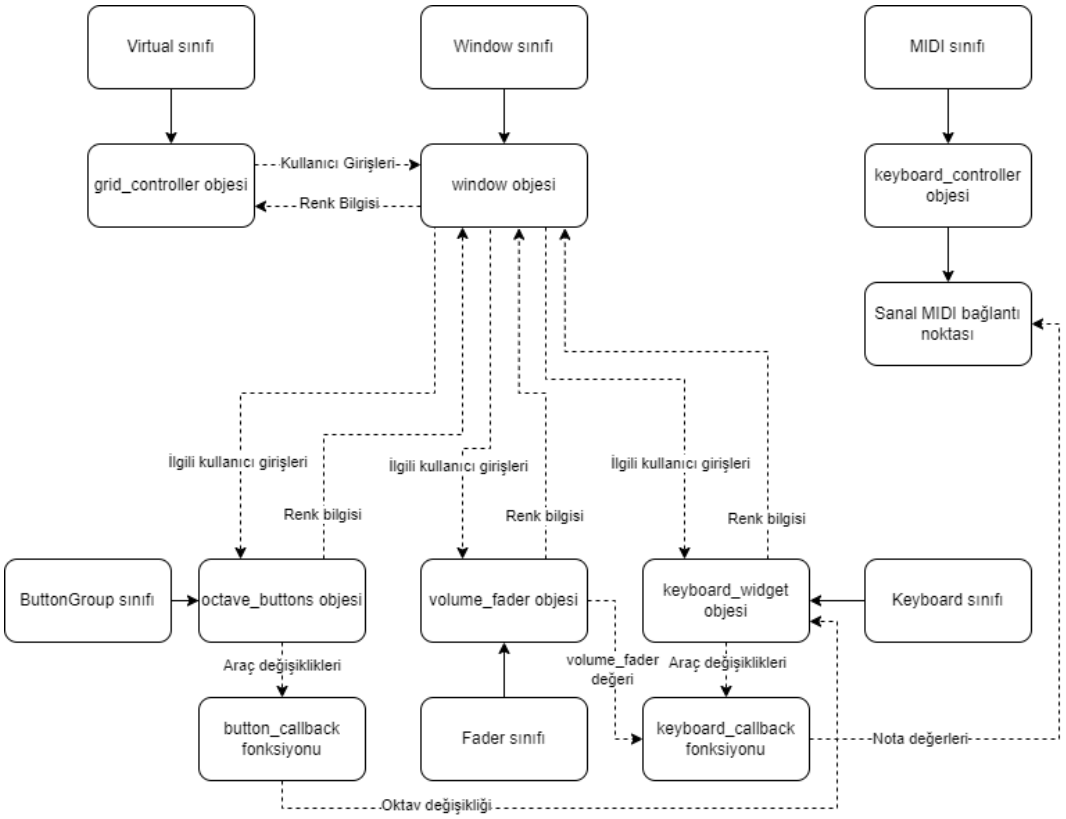
3.3. PyxelWidgets Çatısının Kullanımı

PyxelWidgets çatısı tek başına işlevsel bir yazılım değildir. PyxelWidgets, başka yazılımların geliştirilmesini mümkün kılacak bir araç olarak tasarlanmıştır. Bu bölümde 3 adet örnek yazılım geliştirilerek çatının kullanım şekli tanıtılacaktır.

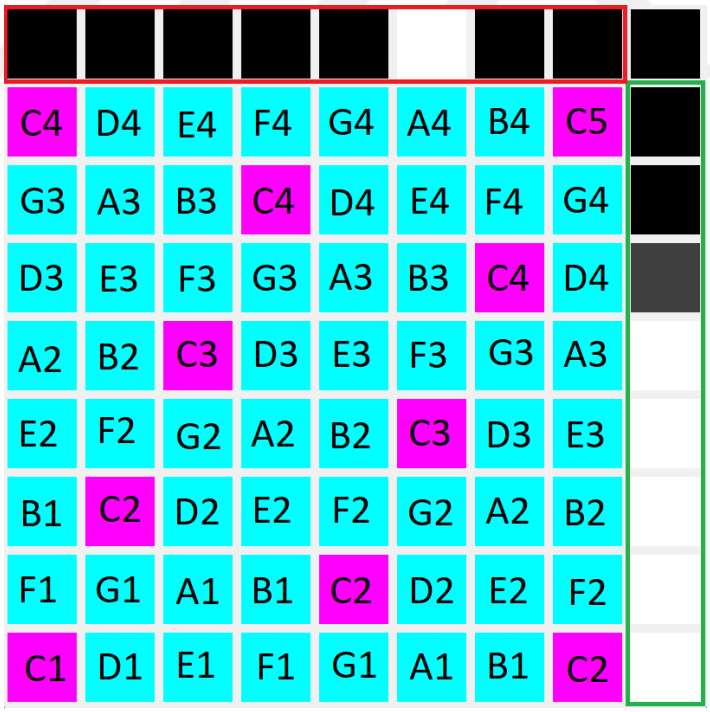
İlk örnekte bir klavye oluşturularak çatının müzik performansı için kullanımı, ikinci örnekte kontrol ögeleri oluşturularak DAW kontrolü için kullanımı ve üçüncü örnekte hem müzik performansı ögelerinin hem de kontrol ögelerinin bulunduğu bir kullanım şekli tanıtılacaktır.

3.3.1. PyxelWidgets çatısı ile klavye örneği

Bu örnekte 9 hücre yüksekliğe ve 9 hücre genişliğe sahip bir sanal kontrolcü ve Window objesi oluşturulmuştur. Window objesine bir Keyboard aracı, bir Fader aracı ve bir ButtonGroup aracı eklenmiştir. Keyboard aracı DiatonicVertical modunda, Do kök notasında ve 4. notadan katlanacak şekilde oluşturulmuştur. Keyboard aracının oluşturduğu notaların sanal müzik enstrümanına gönderilebilmesi için MIDI sınıfından bir kontrolcü daha oluşturularak bu kontrolcü ile sanal bir MIDI bağlantı noktası oluşturulmuştur. Bu örnekte ButtonGroup aracı Keyboard aracının oktavını, Fader aracı ise Keyboard aracının MIDI Note-On mesajı gönderirken kullandığı velocity değerini belirlemektedir. Örneğin kaynak kodu Ek A'da sunulmuş, çalışma şekli Şekil 3.4'de, kontrolcünün açıklamalı görüntüsü Şekil 3.5'te gösterilmiştir. Şekil 3.5'te yeşil alan Fader aracının sınırlarını, turuncu alan ButtonGroup aracının sınırlarını, Keyboard aracı üzerindeki ifadeler ise klavyenin notalarını göstermektedir.



Şekil 3.4 Klavye örneğinin çalışma diyagramı



Şekil 3.5 Klavye örneğinin müzikal ızgara kontrolcü üzerindeki görüntüsü

3.3.2. PyxelWidgets çatısı ile DAW kontrol örneği

Bu örnekte 9 hücre yüksekliğe ve 9 hücre genişliğe sahip bir sanal müzikal ızgara kontrolcüsü ve Window objesi oluşturulmuştur. Window objesine ses yüksekliğini kontrol etmek için yan yana 8 adet Fader aracı; kayıt başlatma, oynatma ve durdurma fonksiyonları için üst üste 3 adet Button aracı; ses kanallarını susturma (mute) fonksiyonu için 8 adet Button aracı eklenmiştir. Örneğin kaynak kodu Ek B'de sunulmuş ve kontrolcü üzerindeki görüntüsü Şekil 3.6'da gösterilmiştir.

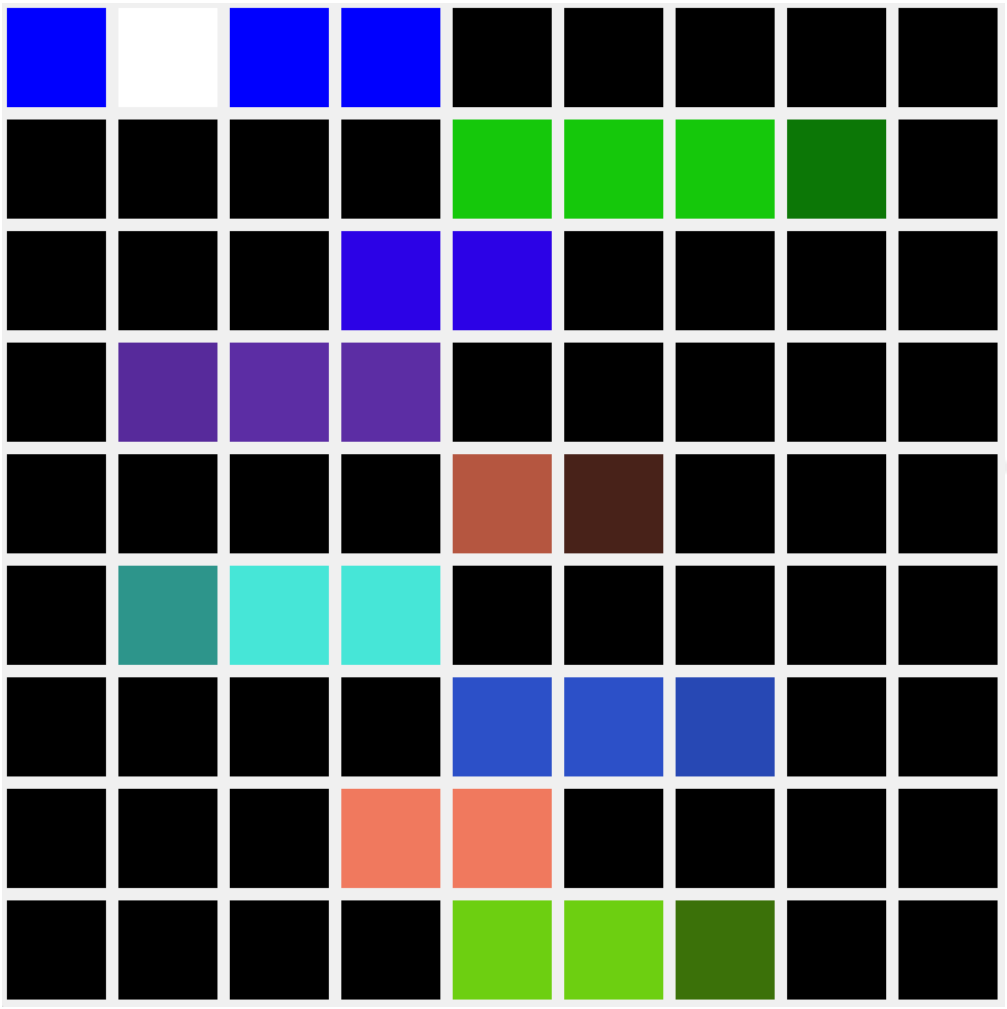
Şekil 3.6 DAW kontrol örneğinin müzikal ızgara kontrolcü üzerindeki görüntüsü

Bu örnekte DAW programının kontrol edilebilmesi için logic control (Mackie Designs Inc., 2002) MIDI mesajlaşma protokolü kullanılmıştır ve Reaper (Cockos Incorporated, 2022) DAW programı ile test edilmiştir.

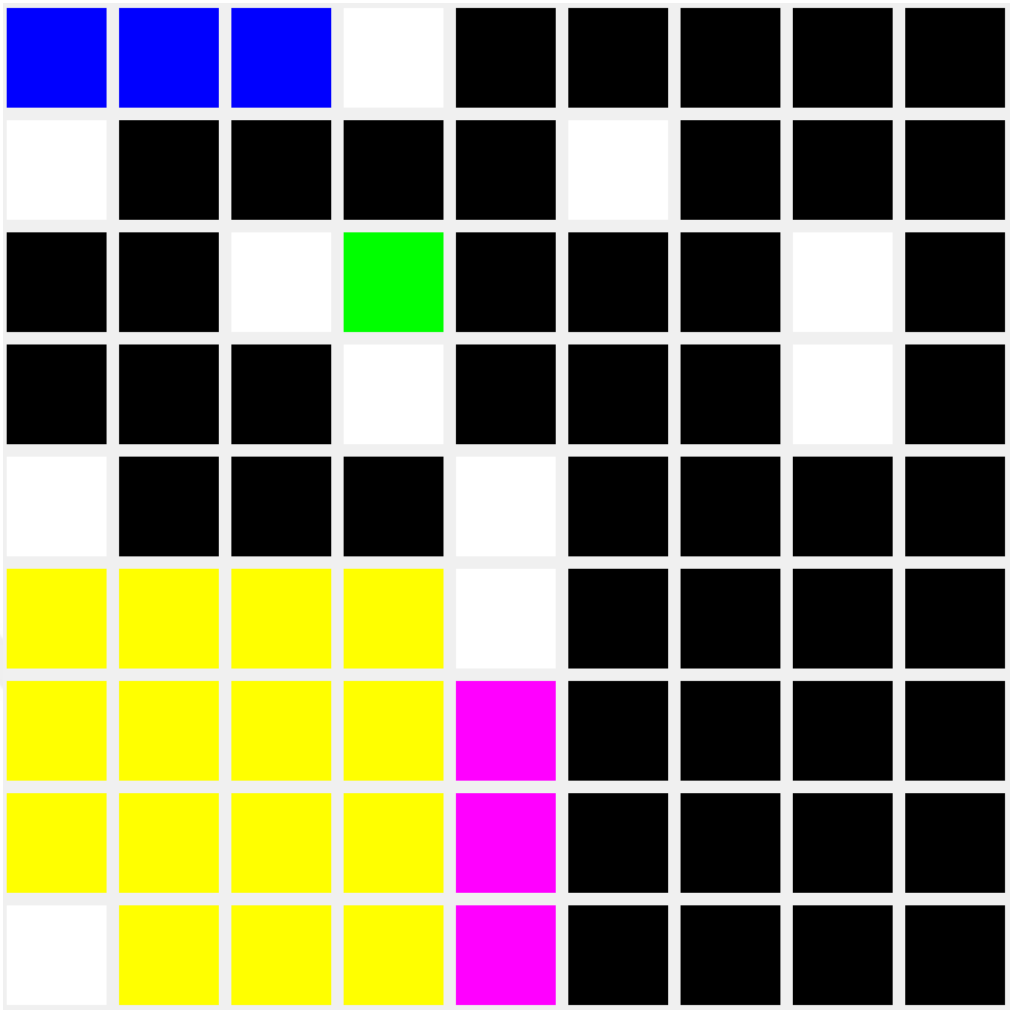
3.3.3. PyxelWidgets çatısı ile çoklu pencere örneği

Bu örnekte 9 hücre yüksekliğe ve 9 hücre genişliğe sahip bir sanal müzikal ızgara kontrolcüsü, bir adet Manager objesi ve farklı büyüklüklerde 6 adet pencere

oluşturulmuştur. Bu pencerelerden biri müzikal ızgara kontrolcüsünün en üst satırındaki hücrelere yerleştirilerek pencereler arası geçiş yapmayı sağlamaktadır. Diğer 5 pencere sırayla Fader araçlarının bulunduğu pencere, Knob araçlarının bulunduğu pencere, Keyboard aracının bulunduğu pencere, Keyboard aracının özelliklerinin değiştirilmesini sağlayan farklı tipte araçlara sahip olan pencere ve Sequencer araçlarının olduğu penceredir. Bu pencerelere yerleştirilmiş araçlar sanal bir MIDI bağlantı noktası üzerinden MIDI Control Change, Note-On ve Note-Off mesajları göndermektedir. Bu sayede hem müzik performansı ögeleri hem de kontrol ögelerinin aynı anda kullanımı gösterilmeye çalışılmıştır.



Şekil 3.7 Çoklu pencere örneğinde Knob penceresinin görüntüsü



Şekil 3.8 Çoklu pencere örneğinde Sequencer penceresinin görüntüsü

Şekil 3.7'de Knob penceresinin müzikal ızgara kontrolcü üzerindeki görüntüsü verilmiştir. Bu pencerede 8 adet Knob aracı üst üste eklenmiştir. Şekil 3.8'de ise Sequencer penceresinin müzikal ızgara kontrolcü üzerindeki görüntüsü verilmiştir. Bu pencerede sol alttaki sarı butonlar aracılığıyla seçilebilecek 16 adet Sequencer aracı bulunmaktadır.

4. SONUÇ, TARTIŞMA VE ÖNERİLER

Bu çalışmada özellikle elektronik müzik üretiminde yaygın olarak kullanılan müzikal ızgara kontrolcüler için PyxelWidgets adlı yazılım çatısı geliştirilmesi hedeflenmiştir. PyxelWidgets'ın geliştirilmesi için öğrenmesi kolay ve yazılım ile ilgilenen müzisyenlerin de kullanabileceği Python programlama dili tercih edilmiştir. Müzikal ızgara kontrolcülerin kullanıcı girişlerinin ve görsel geri bildirim sistemlerinin neredeyse her cihaz için farklı olması sebebiyle, bu cihazların tamamının desteklenebileceği bir mimari tasarlanmaya çalışılmıştır. Ayrıca çalışmanın süresi boyunca erişim sağlanan bütün müzikal ızgara kontrolcülerin desteği de bu çatıya eklenmiştir. İlaveten müzikal ızgara kontrolcüler için arayüz hazırlarken ihtiyaç duyulabilecek ek araçlar da bu çatı ile birlikte geliştirilmiştir.

MIT lisansının kullanıcıların bu yazılımı modifiye etmesine, çoğaltmasına, paylaşmasına ve başka yazılımlar içerisinde kullanılmasına olanak vermesi nedeniyle PyxelWidgets çatısı açık kaynaklı ve MIT lisansına sahip olarak GitHub platformu üzerinden yayınlanmıştır (https://github.com/NullMember/PyxelWidgets). Bu sayede başka kişiler tarafından bu çalışma esnasında erişilemeyen müzikal ızgara kontrolcülerin desteğinin eklenmesi, yazılım geliştirmeleri, hata ayıklama ve yeni özelliklerin eklenmesinin önü açılmıştır.

PyxelWidgets çatısı açık kaynaklı olmasından, uyarlanabilir ve genişletilebilir bir yapıya sahip olmasından, güncel programlama mimarileri ile paralellik göstermesinden ve yeni teknolojilere açık olmasından dolayı hem yazılım geliştiriciler için hem de kullanıcılar için çeşitli avantajlara sahiptir.

Yazılım geliştirme perspektifinden bakıldığında, PyxelWidgets'ın müzikal amaçla üretilmemiş düşük çözünürlüklü ekranlar ile de çalışmaya uygun bir yapıda olması önemli artılarındandır. Böylelikle, örneğin bütün tuşlarının altında renk değerleri değiştirilebilen LED'ler bulunduran bilgisayar klavyeleri de PyxelWidgets çatısı ile uygun sınıfların yazılması durumunda kullanılabilecektir.

PyxelWidgets'in bir diğer avantajı, çatı olarak geliştirilmiş olması sebebiyle sahip olduğu grafiksel araçların yanında yeni araçların eklenebilmesi olanağıdır. Hatta bu araçlar eklenirken çatının kaynak kodu üzerinde bir değişiklik yapılması gerekmemektedir.

PyxelWidgets çatısı ile birlikte birden çok müzikal ızgara kontrolcünün bir arada kullanılabileceği bir pencere yöneticisi de geliştirilmiştir. Pencere yöneticisi, pencere üzerine eklenen araçların birden çok ızgara kontrolcü üzerinde gösterilebilmesini sağlamaktadır. Bu sayede kullanıcı birden çok ızgara kontrolcüye sahip ise bu kontrolcüleri tek bir cihaz gibi kullanabilmektedir.

PyxelWidgets'a kullanıcı gözüyle bakıldığında da birçok önemli katkı sağladığını söylemek mümkündür. Daha açık bir ifadeyle, PyxelWidgets ile ses üretimi ya da tasarımı sırasında mevcut bulunan müzikal ızgara kontrolcülerinin kapasiteleri artmakta ve kullanım kolaylıkları oluşmaktadır. Bu nedenle tasarımcı ya da üretici istediği sonuca daha kolay ulaşabilmektedir. Ayrıca PyxelWidgets yoluyla DAW kullanımı, sentezleyici kullanımı ya da icra sırasında oluşan kısayollar veya benzeri işlem tanımları sayesinde, kullanıcının üretim sürecine olan konsantrasyonu artıracaktır. Bu durum yaratıcı süreci destekleyecek niteliktedir.

PyxelWidgets'ın bütün bu olumlu yönlerinin yanı sıra, bazı olumsuz ya da geliştirilmesi gereken tarafları da vardır. Bunlardan ilki, PyxelWidgets'ın bir çatı olması sebebiyle bu çatıyı kullanacak geliştiricilere çatının özelliklerinin iyi açıklanması gerekliliğidir. PyxelWidgets çatısının kaynak kodunun, bu çalışma için ayrılan süre içerisinde geliştirilmiş olması sebebiyle, zaman sınırlılıklarından dolayı neredeyse hiç dokümante edilmemiştir. Bu sebeple diğer geliştiriciler tarafından anlaşılması zor olacaktır.

PyxelWidgets çatısı düşük işlem gücüne sahip bilgisayarlarda performans sorununa yol açması da çözülmesi gereken problemlerden biridir. Bu sorunun olası iki sebebi Python programlama dilinin yorumlanan bir programlama dili olması ve PyxelWidgets çatısının birden çok işlemci çekirdeği kullanılarak çalışacak şekilde programlanmamış olmasıdır.

Bahsedilmesi gereken bir diğer husus da çatının kullanılabilmesi için kullanıcının Python programlama dilini biliyor olması gerekmektedir. Bu çatı ile birlikte geliştirilebilecek bir grafiksel tasarım aracı kullanıcının Python programlama dilini bilmese dâhi özel grafiksel arayüzler hazırlamasına yardımcı olacaktır. Bu araç, karmaşık fonksiyonların oluşturulan arayüz aracılığıyla tetiklenmesini engelleyecek olsa da basit arayüzler için kullanılabilecektir.

PyxelWidgets'ın geliştirilebilecek bir diğer yönü de bütün işletim sistemleriyle eş biçimde kullanılabilirliğidir. Daha açık bir anlatımla, PyxelWidgets çatısı kullanılarak geliştirilen bir programın araçlardan aldığı veriyi işledikten sonra dışarıya aktarabilmesi gerekmektedir. DAW yazılımları ve sanal enstrümanlar müzikal bilginin iletilmesi için çoğunlukla MIDI iletişim protokolünü kullanmaktadırlar. Bu sebeple aynı bilgisayarda çalışan iki MIDI iletişim protokolü kullanan uygulamanın birbiriyle haberleşmesi için sanal bir MIDI bağlantı noktası oluşturulması gerekmektedir. Linux ve macOS işletim sistemlerinde sanal MIDI bağlantı noktasının oluşturulması harici bir yazılım kullanmaksızın mümkündür. Ancak Windows işletim sisteminde üçüncü parti bir sürücünün kullanılması gerekmektedir. Bu sebeple PyxelWidgets çatısı sanal MIDI bağlantı noktaları oluşturulması gerektiğinde üçüncü parti bir sürücü olan virtualMIDI sürücüsüne bağımlı durumdadır. Bu bağımlılık kullanıcının ek bir program kurması zorunluluğunu doğurmaktadır. PyxelWidgets ile birlikte geliştirilecek bir sanal MIDI bağlantı noktası sürücüsü ile kullanıcı bu bağımlılıktan kurtarılabilir.

PyxelWidgets çatısı yorumlanan bir dil olan Python programlama dili ile geliştirilmiştir. Yorumlanan dillerin sistem kaynaklarını daha çok tüketmesi sebebi ile daha az sistem kaynağına ihtiyaç duyan Rust veya C++ gibi nesne yönelimli programlamayı destekleyen ve derlenen programlama dillerinden biri ile çatının tekrar yazılması mümkündür. C++ ile tekrar yazılması durumunda kullanım kolaylığından feragat edileceği için Rust programlama dilinin daha uygun olacağı düşünülmektedir.

KAYNAKÇA

Alternate Mode Inc. (t.y.). Triggers: DrumKAT Hybrid - Alternate Mode, Inc. 7 Temmuz 2022 tarihinde https://www.alternatemode.com/alternate-mode-electronic-drum-and-percussion-controllers/drumkat-hybrid/ adresinden erişildi.

Anderton, C. (2013, 6 Aralık). A Brief History of MIDI. *Harmony Central*. 19 Mayıs 2022 tarihinde https://www.harmonycentral.com/articles/modules-and-midi/a-brief-history-of-midi-r304/ adresinden erişildi.

Arndt, C. (2022). *SpotlightKid/python-rtmidi*. Python. https://github.com/SpotlightKid/python-rtmidi adresinden erişildi.

Haken, L., Tellman, E. ve Wolfe, P. (1998). An Indiscrete Music Keyboard. *Computer Music Journal*, *22*(1), 30. doi:10.2307/3681043

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362. doi:10.1038/s41586-020-2649-2

Henri, D. (2022). *Hdavid/Launchpad95*. Python. https://github.com/hdavid/Launchpad95 adresinden erişildi.

Hoggenmueller, M., Tomitsch, M. ve Wiethoff, A. (2018). Understanding Artefact and Process Challenges for Designing Low-Res Lighting Displays. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* içinde (ss. 1-12). CHI '18: CHI Conference on Human Factors in Computing Systems, sunulmuş bildiri, Montreal QC Canada: ACM. doi:10.1145/3173574.3173833

Huber, D. M. (2020). *The Midi Manual: A Practical Guide to MIDI within Modern Music Production* (4. bs.). Fourth edition. | New York: Taylor and Francis, 2021. | Series: Audio Engineering Society presents: Routledge. doi:10.4324/9781315670836

inMusicBrands LLC. (t.y.-a). Alesis DM10 MKII Pro Kit. 7 Temmuz 2022 tarihinde https://alesis.com/products/view/dm10-mkii-pro-kit adresinden erişildi.

inMusicBrands LLC. (t.y.-b). APC40 mkII. 20 Temmuz 2022 tarihinde https://www.akaipro.com/apc40-mkii adresinden erişildi.

Jordà, S. (2020). New Musical Interfaces and New Music-making Paradigms. doi:10.48550/ARXIV.2010.01579

Kaisler, S. H. (2005). *Software Paradigms*. Hoboken, N.J: Wiley-Interscience.

KORG Inc. (t.y.). SQ-64—POLY SEQUENCER | KORG (Canada—EN). *KORG Global*. 8 Temmuz 2022 tarihinde https://www.korg.com/caen/products/dj/sq_64/ adresinden erişildi.

Lehrman, P. D. ve Tully, T. (1993). *MIDI for the Professional* (Revised and updated ed.). New York , NY: Amsco Publ.

Loud Audio LLC. (t.y.). MCU Pro Universal Control | Controllers | MACKIE. 11 Temmuz 2022 tarihinde https://mackie.com/en/products/controllers/mcu-pro-and-xt-pro/mcu_pro.html adresinden erişildi.

Mann, S. (2007). Natural Interfaces for Musical Expression: Physiphones and a physics-based organology. *Proceedings of the 7th international conference on New interfaces for musical expression—NIME '07* içinde (s. 118). NIME, sunulmuş bildiri, New York, New York: ACM Press. doi:10.1145/1279740.1279761

Martinez, W. L. (2011). Graphical user interfaces. *Wiley Interdisciplinary Reviews: Computational Statistics*, *3*(2), 119-133. doi:10.1002/wics.150

M-Audio. (t.y.). Oxygen Pro 61 | M-Audio. 19 Temmuz 2022 tarihinde https://www.m-audio.com/oxygen-pro-61 adresinden erişildi.

MIDI Manufacturers Association Incorporated. (t.y.). MIDI Time Code. The MIDI Manufacturers Association. https://www.midi.org/specifications/midi-time-code/download adresinden erişildi.

MIDI Manufacturers Association Incorporated ve Japan MIDI Standards Committee. (1996, Şubat). MIDI 1.0 Detailed Specification. The MIDI Manufacturers Association. https://midi.org/specifications/m1-v4-2-1-midi-1-0-detailed-specification-96-1-4/download adresinden erişildi.

Music Tribe. (t.y.). Behringer | Product | X-TOUCH. 11 Temmuz 2022 tarihinde https://www.behringer.com/product.html?modelCode=P0B1X adresinden erişildi.

Native Instruments GmbH. (t.y.). Keyboards: Komplete Kontrol S49 / S61 | Komplete. 19 Temmuz 2022 tarihinde https://www.native-instruments.com/en/products/komplete/keyboards/komplete-kontrol-s49-s61/ adresinden erişildi.

Novation Digital Music Systems Ltd. (t.y.-a). Components—Launchpad Mini MK3, Launchpad X, and Launchpad Pro MK3 Custom Mode Editor Guide. *Novation*. 15 Temmuz 2022 tarihinde https://support.novationmusic.com/hc/en-gb/articles/360009860380--Components-Launchpad-Mini-MK3-Launchpad-X-and-Launchpad-Pro-MK3-Custom-Mode-Editor-Guide adresinden erişildi.

Novation Digital Music Systems Ltd. (t.y.-b). Launch | Novation. 20 Temmuz 2022 tarihinde https://novationmusic.com/en/launch adresinden erişildi.

Paradiso, J. A. ve O'Modhrain, S. (2003). Current Trends in Electronic Music Interfaces. Guest Editors' Introduction. *Journal of New Music Research, 32*(4), 345-349. doi:10.1076/jnmr.32.4.345.18855

Perrier, M. (t.y.). Arturia—BeatStep PRO - BeatStep Pro. 8 Temmuz 2022 tarihinde https://www.arturia.com/products/hybrid-synths/beatstep-pro/overview adresinden erişildi.

PreSonus Audio Electronics Inc. (t.y.). FaderPort 16. *PreSonus*. 11 Temmuz 2022 tarihinde https://www.presonus.com/products/FaderPort-16 adresinden erişildi.

Reimer, J. (2005, 5 Mayıs). A History of the GUI. *Ars Technica*. 20 Temmuz 2022 tarihinde https://arstechnica.com/features/2005/05/gui/ adresinden erişildi.

Rossmy, B. ve Wiethoff, A. (2021). Musical Grid Interfaces: Past, Present, and Future Directions. *NIME 2021* içinde . NIME 2021, sunulmuş bildiri, Shanghai, China: PubPub. doi:10.21428/92fbeb44.6a2451e6

Rossum, G. van ve Drake, F. L. (2010). *The Python language reference*. Python documentation manual / Guido van Rossum; Fred L. Drake [ed.] (Release 3.0.1 [Repr.].). Hampton, NH: Python Software Foundation.

Simula - Guide: History, Origin, and More. (2021, 4 Ocak).*History-Computer*. https://history-computer.com/simula-guide/ adresinden erişildi.

Smith, D. ve Wood, C. (1981). The "USI", or Universal Synthesizer Interface. *Audio Engineering Society Convention 70* içinde . http://www.aes.org/e-lib/browse.cfm?elib=11909 adresinden erişildi.

The MIDI Association. (t.y.). MIDI History:Chapter 6-MIDI Is Born 1980-1983. *The MIDI Association*. 19 Mayıs 2022 tarihinde https://www.midi.org/midi-articles/midi-history-chapter-6-midi-is-born-1980-1983 adresinden erişildi.

Vallis, O., Hochenbaum, J. ve Kapur, A. (2010). A Shift Towards Iterative And Open-Source Design For Musical Interfaces. doi:10.5281/ZENODO.1177918

Vincent, J. (2015, 23 Eylül). Feeling the music with Roli's squishy, pressure-sensitive keyboard. *The Verge*. 7 Temmuz 2022 tarihinde https://www.theverge.com/2015/9/23/9372961/roli-seaboard-rise-grand-hands-on adresinden erişildi.

Weisfeld, M. A. (2013). *The Object-Oriented Thought Process* (Fourth edition.). Upper Saddle River, NJ: Addison-Wesley.

EKLER

Ek A, PyxelWidgets çatısı ile klavye örneği

```
from PyxelWidgets.Controllers.Virtual import Virtual
from PyxelWidgets.Controllers.MIDI import MIDI
from PyxelWidgets.Utils.Enums import Event
from PyxelWidgets.Window import Window
from PyxelWidgets.Widgets import *
import time

def keyboard_callback(name, event, data):
    if event == Event.Changed:
        note, velocity = data
        if velocity > 0.0:
            keyboard_controller.sendNoteOn(note, int(velocity * volume_fader.value *
127.0))
        else:
            keyboard_controller.sendNoteOff(note)

def button_callback(name, event, data):
    if event == Event.Pressed:
        x, y = data
        keyboard_widget.octave = x

window = Window(9, 9)

keyboard_widget = Keyboard.Keyboard(
    x = 0,
    y = 0,
    width = 8,
    height = 8,
    type = Keyboard.Keyboard.Type.DiatonicVertical,
    scale = Keyboard.Keyboard.Scale.Major,
    root = Keyboard.Keyboard.Root.C,
    callback = keyboard_callback)
octave_buttons = ButtonGroup.ButtonGroup(0, 8, 8, 1, callback = button_callback)
octave_buttons.tapped(5, 0)
volume_fader = Fader.Fader(8, 0, 1, 8)
volume_fader.tapped(0, 5)

window.addWidget(keyboard_widget)
window.addWidget(octave_buttons)
window.addWidget(volume_fader)

grid_controller = Virtual(9, 9)
grid_controller.init()
grid_controller.connect()
grid_controller.setCallback(window.process)

keyboard_controller = MIDI()
```

Ek A (devam), PyxelWidgets çatısı ile klavye örneği

```
keyboard_controller.init()
keyboard_controller.connectVirtual("Keyboard")

while True:
    try:
        grid_controller.update(window.update())
        time.sleep(1 / 60)
    except KeyboardInterrupt:
        grid_controller.close()
        break
```

Ek B, PyxelWidgets çatısı ile DAW kontrol örneği

```
from PyxelWidgets.Controllers.Virtual import Virtual
from PyxelWidgets.Controllers.MIDI import MIDI
from PyxelWidgets.Utils.Enums import Event
from PyxelWidgets.Utils.Pixel import Pixel, Colors
from PyxelWidgets.Window import Window
from PyxelWidgets.Widgets import *
import time
import random
import enum

class DAW(MIDI):
    class Buttons(enum.Enum):
        MUTE_0 = 16
        MUTE_1 = 17
        MUTE_2 = 18
        MUTE_3 = 19
        MUTE_4 = 20
        MUTE_5 = 21
        MUTE_6 = 22
        MUTE_7 = 23
        STOP = 93
        PLAY = 94
        RECORD = 95

    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def processMIDI(self, message, _):
        data, delta = message
        cmd = data[0] & 0xF0
        chn = data[0] & 0x0F
        if cmd == 0x90: # MCU Leds
            button = data[1]
            value = data[2]
                        if button >= DAW.Buttons.MUTE_0.value and button <
DAW.Buttons.MUTE_7.value: # MUTE1-8
                window.widgets[f"Mute_{button - 16}"].value = 1.0 if value else 0.0
            elif button == DAW.Buttons.STOP.value: # STOP
                window.widgets["Stop"].value = 1.0 if value else 0.0
            elif button == DAW.Buttons.PLAY.value: # PLAY
                window.widgets["Play"].value = 1.0 if value else 0.0
            elif button == DAW.Buttons.RECORD.value: # RECORD
                window.widgets["Record"].value = 1.0 if value else 0.0
        if cmd == 0xE0: # MCU Faders
            if chn >= 0 and chn < 8: # Channel Faders
                value = ((data[2] << 7) | data[1]) / (1 << 14)
                window.widgets[f"Fader_{chn}"].value = value
```

Ek B (devam), PyxelWidgets çatısı ile DAW kontrol örneği

```
def fader_callback(name, event, data):
    name, index = name.split("_")
    index = int(index)
    if event == Event.Changed:
        daw_controller.sendPitchBend(int(data * 16383.0), index)

def button_callback(name, event, data):
    if event == Event.Changed:
        if "Mute" in name:
            name, index = name.split("_")
            index = int(index)
            daw_controller.sendNoteOn(DAW.Buttons.MUTE_0.value + index, 127)
            daw_controller.sendNoteOff(DAW.Buttons.MUTE_0.value + index, 0)
        if name == "Stop":
            daw_controller.sendNoteOn(DAW.Buttons.STOP.value, 127)
            daw_controller.sendNoteOff(DAW.Buttons.STOP.value, 0)
        if name == "Play":
            daw_controller.sendNoteOn(DAW.Buttons.PLAY.value, 127 if data else 0)
            daw_controller.sendNoteOff(DAW.Buttons.PLAY.value, 0)
        if name == "Record":
             daw_controller.sendNoteOn(DAW.Buttons.RECORD.value, 127 if data else
0)
            daw_controller.sendNoteOff(DAW.Buttons.RECORD.value, 0)

window = Window(9, 9)
for i in range(8):
    window.addWidget(
        Fader.Fader(
            x = i, y = 0, width = 1, height = 8,
            name = f"Fader_{i}",
            activeColor = Pixel(
                r = random.randint(0, 255),
                g = random.randint(0, 255),
                b = random.randint(0, 255)
            ),
            lock = True, callback = fader_callback
        )
    )

for i in range(8):
    window.addWidget(
        Button.Button(
            x = i, y = 8, width = 1, height = 1,
            name = f"Mute_{i}", mode = Button.Button.Mode.Switch,
            activeColor = Colors.Red,
            deactiveColor = Colors.DarkRed,
            lock = True, callback = button_callback)
```

Ek B (devam), PyxelWidgets çatısı ile DAW kontrol örneği

```
    )

window.addWidget(Button.Button(
    x = 8, y = 0, width = 1, height = 1,
    name = "Record", callback = button_callback, lock = True,
    activeColor = Colors.Red, deactiveColor = Colors.Red * 0.5)
)
window.addWidget(Button.Button(
    x = 8, y = 1, width = 1, height = 1,
    name = "Play", callback = button_callback, lock = True,
    activeColor = Colors.Green, deactiveColor = Colors.Green * 0.5)
)
window.addWidget(Button.Button(
    x = 8, y = 2, width = 1, height = 1,
    name = "Stop", callback = button_callback, lock = True,
    activeColor = Colors.Yellow, deactiveColor = Colors.Yellow * 0.5)
)

grid_controller = Virtual(9, 9)
grid_controller.init()
grid_controller.connect()
grid_controller.setCallback(window.process)

daw_controller = DAW()
daw_controller.init()
daw_controller.connectVirtual("DAW")

while True:
    try:
        grid_controller.update(window.update())
        time.sleep(1 / 60)
    except KeyboardInterrupt:
        grid_controller.close()
        daw_controller.close()
        break
```

Ek C, PyxelWidgets çatısı ile çoklu pencere örneği

```
from PyxelWidgets.Controllers.Virtual import Virtual
from PyxelWidgets.Controllers.MIDI import MIDI
from PyxelWidgets.Window import Window, Manager
from PyxelWidgets.Utils.Enums import *
from PyxelWidgets.Utils.Rectangle import *
from PyxelWidgets.Utils.Pixel import *
from PyxelWidgets.Widgets import *
import time
import random

def buttonCallback(name: str, event: str, data):
    global currentWindow
    x, y = data
    if event == Event.Pressed:
        if windows[currentWindow] == sequencerWindow:
            sequencerWindow.removeWidget(sequencerWidgets[currentSequencer].nam
e)
        if windows[currentWindow] == keyboardWindow:
            manager.removeWindow(keyboardConfigWindow.name)
        manager.removeWindow(windows[currentWindow].name)
        currentWindow = x
        manager.addWindow(windows[currentWindow], 0, 0, 9, 8)
        if windows[currentWindow] == sequencerWindow:
            sequencerWindow.addWidget(sequencerWidgets[currentSequencer])

def faderCallback(name: str, event: str, data):
    if event == Event.Changed:
        index = name.split("_")[1]
        index = int(index)
        keyboard.sendControlChange(index + 8, int(data * 127.0))

def knobCallback(name: str, event: str, data):
    if event == Event.Changed:
        index = name.split("_")[1]
        index = int(index)
        keyboard.sendControlChange(index + 16, int(data * 127.0))

def keyboardCallback(name: str, event: str, data):
    if name == "Keyboard":
        if event == Event.Changed:
            note, velocity = data
            keyboard.sendNoteOn(note, int(velocity * 127.0))
    if name == "switch":
        if event == Event.Released:
            manager.removeWindow(keyboardWindow.name)
            manager.addWindow(keyboardConfigWindow, 0, 0, 9, 8)
```

Ek C (devam), PyxelWidgets çatısı ile çoklu pencere örneği

```
def keyboardConfigCallback(name: str, event: str, data):
    keyboard = keyboardWindow.widgets['Keyboard']
    if name == "Type":
        if event == Event.Pressed:
            x, y = data
            keyboard.type = Keyboard.Keyboard.Type(x)
    if name == "Root":
        if event == Event.Changed:
            note, velocity = data
            note %= 12
            if note < len(Keyboard.Keyboard.Root):
                keyboard.root = Keyboard.Keyboard.Root(note)
    elif name == "Octave":
        if event == Event.Pressed:
            x, y = data
            keyboard.octave = x
    elif name == "Scale":
        if event == Event.Pressed:
            scale = keyboardConfigWindow.widgets['Scale']
            x, y = data
            scale = x + (y * scale.width)
            if scale < len(Keyboard.Keyboard.Scale):
                keyboard.scale = Keyboard.Keyboard.Scale(scale)
    elif name == "switch":
        if event == Event.Released:
            manager.removeWindow(keyboardConfigWindow.name)
            manager.addWindow(keyboardWindow, 0, 0, 9, 8)

def sequencerCallback(name: str, event: str, data):
    global currentSequencer
    if "Sequencer" in name:
        prefix, index = name.split("_")
        index = int(index)
        if event == Event.Tick:
            keyboard.sendNoteOn(36 + index, 0, 1)
        if event == Event.Active:
            keyboard.sendNoteOn(36 + index, 100 + random.randint(-10, 10), 1)
    elif "sample" in name:
        if event == Event.Pressed:
            x, y = data
             sequencerWindow.removeWidget(sequencerWidgets[currentSequencer].nam
e)
            currentSequencer = x + (y * 4)
            keyboard.sendNoteOn(36 + currentSequencer, 100, 1)
            keyboard.sendNoteOn(36 + currentSequencer, 0, 1)
            sequencerWindow.addWidget(sequencerWidgets[currentSequencer])
        if event == Event.Held:
```

Ek C (devam), PyxelWidgets çatısı ile çoklu pencere örneği

```
            x, y = data
            sequencerWidgets[x + (y * 4)].reset()
    elif "height" in name:
        if event == Event.Pressed:
            x, y = data
            for sequencer in sequencerWidgets:
                sequencer.step = (y + 1) * 8
            sequencerWindow.forceUpdate()

controller = Virtual(9, 9)
controller.init(True)
controller.connect()

keyboard = MIDI()
keyboard.init()
keyboard.connectVirtual('Keyboard')

faderWindow = Window(9, 8)
knobWindow = Window(9, 8)
keyboardWindow = Window(9, 8)
keyboardConfigWindow = Window(9, 8)
sequencerWindow = Window(9, 8)

topButtonWindow = Window(8, 1)

windows = [faderWindow, knobWindow, keyboardWindow, sequencerWindow]
currentWindow = 0

manager = Manager(9, 9)
manager.addWindow(topButtonWindow, 0, 8, 8, 1)
manager.addWindow(windows[currentWindow], 0, 0, 9, 8)
manager.addController(controller, 0, 0)

topButtonWindow.addWidget(
    ButtonGroup.ButtonGroup(
        x = 0, y = 0, width = len(windows), height = 1,
        deactiveColor = Colors.Blue,
        callback = buttonCallback)
)

for i in range(8):
    faderWindow.addWidget(
        Fader.Fader(
            x = i, y = 0, width = 1, height = 8,
            name = f"DAWFader_{i}",
            callback = faderCallback,
```

Ek C (devam), PyxelWidgets çatısı ile çoklu pencere örneği

```
            activeColor = Pixel(random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255)))
    )

for i in range(8):
    knobWindow.addWidget(
        Knob.Knob(
            x = 0, y = 7 - i, width = 8, height = 1,
            name = f"DAWKnob_{i}",
            callback = knobCallback,
            type = Knob.Knob.Type.BoostCut,
            activeColor = Pixel(random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255)))
    )

keyboardWindow.addWidget(
    Keyboard.Keyboard(
        x = 0, y = 0, width = 8, height = 8,
        name = "Keyboard",
        type = Keyboard.Keyboard.Type.DiatonicVertical,
        scale = Keyboard.Keyboard.Scale.Major,
        root = Keyboard.Keyboard.Root.C,
        fold = 4, octave = 4,
        callback = keyboardCallback)
)
keyboardWindow.addWidget(
    Button.Button(
        8, 0, 1, 1,
        name = "switch",
        callback = keyboardCallback,
        deactiveColor = Colors.Red)
)
keyboardConfigWindow.addWidget(
    ButtonGroup.ButtonGroup(
        x = 0, y = 7, width = len(Keyboard.Keyboard.Type), height = 1,
        name = "Type",
        callback = keyboardConfigCallback)
)
keyboardConfigWindow.addWidget(
    Keyboard.Keyboard(
        x = 0, y = 5, width = 8, height = 2,
        name = "Root",
        type = Keyboard.Keyboard.Type.Keyboard,
        callback = keyboardConfigCallback)
)
keyboardConfigWindow.addWidget(
    ButtonGroup.ButtonGroup(
```

Ek C (devam), PyxelWidgets çatısı ile çoklu pencere örneği

```
        x = 0, y = 4, width = 8, height = 1,
        name = "Octave",
        deactiveColor = Colors.PaleVioletRed,
        callback = keyboardConfigCallback)
)
keyboardConfigWindow.addWidget(
    ButtonGroup.ButtonGroup(
        x = 0, y = 0, width = 8, height = 2,
        name = "Scale",
        deactiveColor = Colors.SkyBlue,
        callback = keyboardConfigCallback)
)
keyboardConfigWindow.addWidget(
    Button.Button(
        8, 0, 1, 1,
        name = "switch",
        callback = keyboardConfigCallback,
        deactiveColor = Colors.Red)
)

sequencerWidgets = []
for i in range(16):
    sequencerWidgets.append(
        Sequencer.Sequencer(
            x = 0, y = 4, width = 8, height = 4,
            clock = controller.clock,
            callback = sequencerCallback)
    )

sequencerWindow.addWidget(
    ButtonGroup.ButtonGroup(
        x = 0, y = 0, width = 4, height = 4,
        deactiveColor = Pixel(255, 255, 0),
        callback = sequencerCallback,
        name = "sample")
)
sequencerWindow.addWidget(
    ButtonGroup.ButtonGroup(
        x = 4, y = 0, width = 1, height = 4,
        deactiveColor = Pixel(255, 0, 255),
        callback = sequencerCallback,
        name = "height")
)
currentSequencer = 0

if __name__ == "__main__":
    while True:
```

Ek C (devam), PyxelWidgets çatısı ile çoklu pencere örneği

```
try:
    manager.update()
    time.sleep(1 / 60)
except KeyboardInterrupt:
    manager.destroy()
    break
```